Programming With QuickTime VR 2.1

With Reference Sections

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

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, ADC will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to ADC.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures, Tables, and Listings

About This Book

This book, *Programming With QuickTime VR 2.1*, describes version 2.1 of QuickTime VR, an extension of the QuickTime technology developed by Apple Computer, Inc. that allows users to interactively explore and examine photorealistic, three-dimensional virtual worlds.

Primarily, this book describes the QuickTime VR Manager, which is the part of QuickTime 3 that you can use to control QuickTime VR movies from your application. For example, you can use the QuickTime VR Manager to

- display movies of panoramas and objects
- perform basic orientation, positioning, and animation control
- intercept and override QuickTime VR's mouse-tracking and default hot spot behaviors
- composite flat or perspective overlays (such as QuickDraw 3D objects or QuickTime movies)
- specify transition effects
- control QuickTime VR's memory usage
- intercept calls to some QuickTime VR Manager functions and modify their behavior

This book also describes the QuickTime VR file format (the format of the movie files that contain QuickTime VR movies). You need this information only if you need to parse existing QuickTime VR movies or you want to create QuickTime VR movies programmatically. For instance, you need this information if you are developing QuickTime VR movie-authoring software. In general, however, you don't need to know about the format of atoms or atom containers simply to use the functions provided by the QuickTime VR Manager.

Note
This book does not describe how to capture VR scenes or author QuickTime VR movies using tools such as the QuickTime VR Authoring Studio. See the documentation provided with your authoring software for complete information. ◆

Conventions Used in This Book

This book uses special conventions to present certain types of information. Words that require special treatment appear in specific fonts or font styles.

Special Fonts

All code listings, reserved words, and the names of actual data structures, constants, fields, parameters, and routines are shown in Letter Gothic (`this is Letter Gothic`).

Words that appear in **boldface** are key terms or concepts and are defined in the glossary.

Types of Notes

There are two types of notes used in this book.

Note
A note like this contains information that is interesting but possibly not essential to an understanding of the main text. ◆

IMPORTANT
A note like this contains information that is essential for an understanding of the main text. ▲

Development Environment

The system software routines described in this book are available using C interfaces. How you access these routines depends on the development environment you are using. When showing QuickTime VR routines, this book uses the C interfaces available with the Macintosh Programmer's Workshop (MPW).

All code listings in this book are shown in C. They show methods of using various routines and illustrate techniques for accomplishing particular tasks. All code listings have been compiled and tested.

IMPORTANT

For any online updates to this book, check the QuickTime developers' page on the World Wide Web, at

http://www.apple.com/quicktime/developers/index.html

or you may go directly to the documentation page at

http://devworld.apple.com/techinfo/techdocs/multimedia/qtdevdocs/index.htm ▲

For More Information

If you provide commercial products and services, call 408-974-4897 for information on the developer support programs available from Apple.

For information on registering signatures, file types, and other technical information, contact

Macintosh Developer Technical Support
Apple Computer, Inc.
1 Infinite Loop, M/S 303-2T
Cupertino, CA 95014

P R E F A C E

CHAPTER 1

About QuickTime VR

Contents

QuickTime VR is an extension of the QuickTime technology developed by Apple Computer, Inc. that allows users to interactively explore and examine photorealistic, three-dimensional virtual worlds. Unlike many other virtual reality systems, QuickTime VR does not require the user to wear goggles or gloves. Instead, the user navigates in a virtual world using standard input devices (such as the mouse or keyboard) to change the image displayed by the QuickTime VR movie controller.

The images displayed in QuickTime VR movies can be either captured photographically or rendered on a computer using a three-dimensional (3D) graphics package.

Displaying QuickTime VR Movies

QuickTime VR movies are simply a special kind of QuickTime movie, so it's easy to add support to your application for playing QuickTime VR movies. If the QuickTime VR Manager (and hence the QuickTime VR movie controller) is available, you simply open a movie using standard QuickTime functions, call `NewMovieController` to associate the movie with the QuickTime VR movie controller, and make the appropriate call to `MCIsPlayerEvent` in your main event loop. These are exactly the same steps you follow to open and manage any QuickTime movie. Listing 1-1 shows a typical way to open a QuickTime VR movie.

Listing 1-1 Opening a QuickTime VR movie

```
Movie MyGetMovie (void)
{
    OSErr                   myErr;
    SFTypeList              myTypes = {MovieFileType, 0, 0, 0};
    StandardFileReply       myReply;
    Movie                   myMovie = nil;
    short                   myResFile;

    StandardGetFilePreview(nil, 1, myTypes, &myReply);
    if (myReply.sfGood) {
        myErr = OpenMovieFile(&myReply.sfFile, &myResFile, fsRdPerm);
```

```
        if (myErr == noErr) {
            short           myResID = 0;        //We want the first movie.
            Str255          myName;
            Boolean         wasChanged;

            myErr = NewMovieFromFile(&myMovie, myResFile, &myResID, myName,
                                     newMovieActive, &wasChanged);
            CloseMovieFile(myResFile);
        }
    }
    return(myMovie);
}
```

It's important to notice that Listing 1-1 does not use the QuickTime VR Manager at all. Instead, it relies entirely on QuickTime's Movie Toolbox and other Macintosh system software managers.

Note
See *QuickTime 3 Reference* for a complete description of the Movie Toolbox. ◆

Once you've opened a file containing a QuickTime VR movie, you need to call `NewMovieController` to obtain the standard user interface for playing QuickTime VR movies. It's particularly important that you call `NewMovieController` (rather than call the Component Manager directly) for QuickTime VR movies, because QuickTime VR movies contain special information that lets QuickTime know which movie controller to load.

In your main event loop, you should pass all events to the `MCIsPlayerEvent` function, which passes user events (such as mouse movements and button clicks) to the QuickTime VR movie controller. QuickTime VR automatically changes the cursor's shape when it's inside the movie's boundary. As a result, your application should relinquish control of the cursor for as long as it remains in the movie's boundary and then reset the cursor's shape as necessary when it is moved outside the movie.

To allow the QuickTime VR movie controller to update the shape of the cursor in a timely manner, your application should pass all events, even idle events, to the `MCIsPlayerEvent` function. Alternatively, you can call the `MCIdle` function frequently.

If you want to disable the automatic cursor tracking and shape changing provided by the QuickTime VR movie controller, you can execute this line of

code, where `myMC` is an identifier for a movie controller returned by `NewMovieController`:

```
MCDoAction(myMC, mcActionSetCursorSettingEnabled, (void*) false);
```

Note
The `mcActionSetCursorSettingEnabled` movie controller action was introduced in QuickTime version 2.1. See the chapter "QuickTime VR Movie Controller" in this book for a description of how the QuickTime VR movie controller handles this and other movie controller actions. ◆

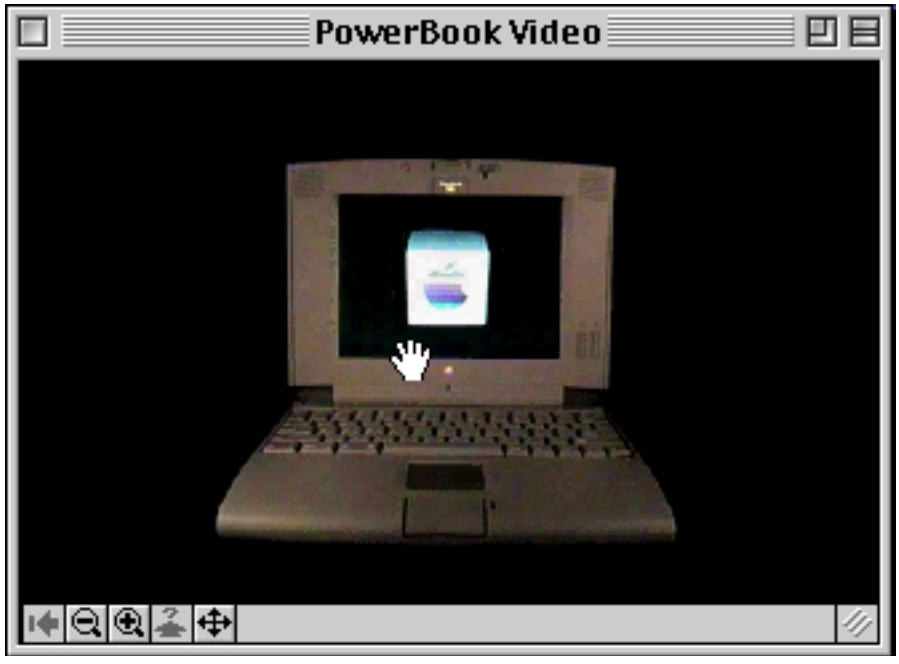
Movies and Nodes

The data for a QuickTime VR virtual world is stored in a QuickTime VR movie. A **QuickTime VR movie** contains a single **scene**, which is a collection of one or more nodes. A **node** is a position in a virtual world at which an object or panorama can be viewed. For a panoramic node, the position of the node is the point from which the panorama is viewed. QuickTime VR scenes can contain any number of nodes, which can be either object or panoramic nodes.

Note
QuickTime uses the term *movie* to emphasize the time-based nature of QuickTime data (such as video and audio data streams). QuickTime VR uses the same term solely on analogy with QuickTime movies; in general, QuickTime VR data is not time-based. ◆

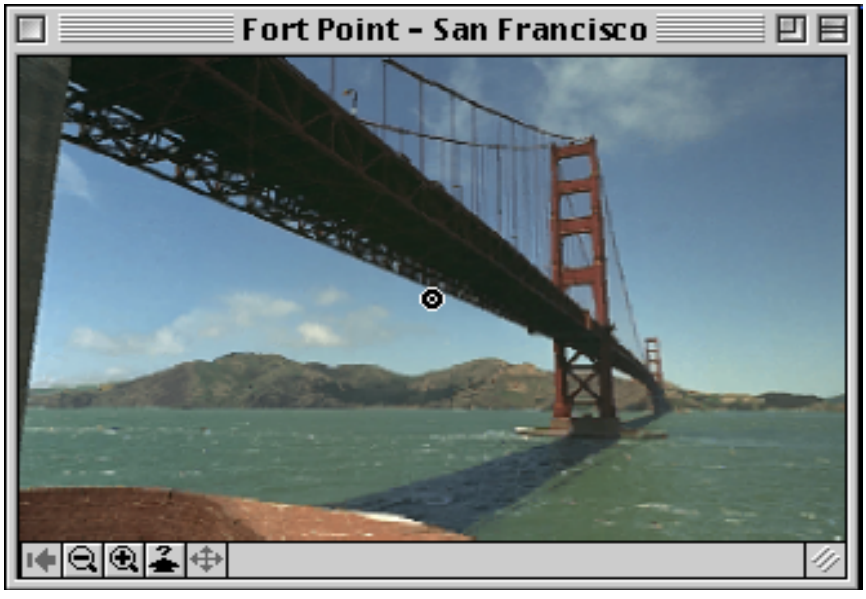
An **object node** (or, more briefly, an **object**) provides a view of a single object or a closely grouped set of objects. You can think of an object node as providing a view of an object from the outside looking in. Figure 1-1 shows one view of an object node. The user can use the mouse or keyboard to change the horizontal and vertical viewing angles to move around the object. The user can also zoom in or out to enlarge or reduce the size of the displayed object. Object nodes are often designed to give the illusion that the user is picking up and turning an object (in Figure 1-1, a Macintosh PowerBook computer) and viewing it from all angles.

Figure 1-1 An object in a QuickTime VR virtual world



A **panoramic node** (or, more briefly, a **panorama**) provides a panoramic view of a particular location, such as you would get by turning around on a rotating stool. You can think of a panoramic node as providing a view of a location from the inside looking out. Figure 1-2 shows one view of a panoramic node. As with object nodes, the user can use the mouse (or keyboard) to navigate in the panorama and to zoom in and out.

Figure 1-2 A panorama in a QuickTime VR virtual world



A node in a QuickTime VR movie is identified by a unique **node ID,** a long integer that is assigned to the node at the time a VR movie is created (and that is stored in the movie file).

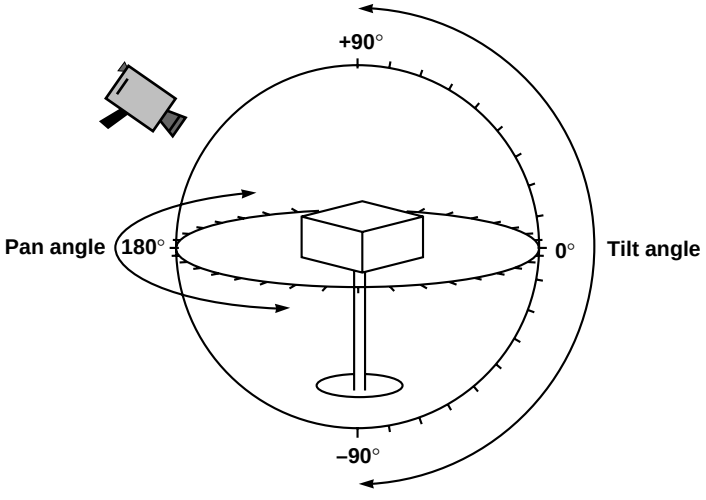
When a QuickTime VR movie contains more than one node, the user can move from one node to another if the author of the QuickTime VR movie has provided a **link** (or connection) between the source and destination nodes. A link between nodes is depicted graphically by a **link hot spot,** a type of hot spot that, when clicked, moves the user from one node in a scene to another node.

Note
It's also possible to move from node to node programmatically, using the QuickTime VR Manager, even between nodes that were not explicitly linked by the movie's author. ◆

Object Nodes

The data used to represent an object is stored in a QuickTime VR movie's video track as a sequence of individual frames, where each frame represents a single view of the object. An **object view** is completely determined by its node ID, field of view, view center, pan angle, tilt angle, view time, and view state. Figure 1-3 illustrates the pan and tilt angles of an object view.

Figure 1-3 Pan and tilt angles of an object



In QuickTime VR, angles can be specified in either radians or degrees. (The default angular unit is degrees.) A view's pan angle typically ranges from 0 degrees to 360 degrees (that is, from 0 to 2π radians). When a user is looking directly at the equator of a multirow object, the tilt angle is 0. Increasing the tilt angle rotates the object down, while decreasing the tilt angle rotates the object up. Setting the tilt angle to 90 degrees results in a view that is looking straight down at the top of the object; setting the tilt angle to –90 degrees results in a view that is looking straight up at the bottom of the object. In general, the normal range for tilt angles is from –90 degrees to +90 degrees. You can, however, set the tilt angle to a value greater than 90 degrees if the movie contains upside-down views of the object.

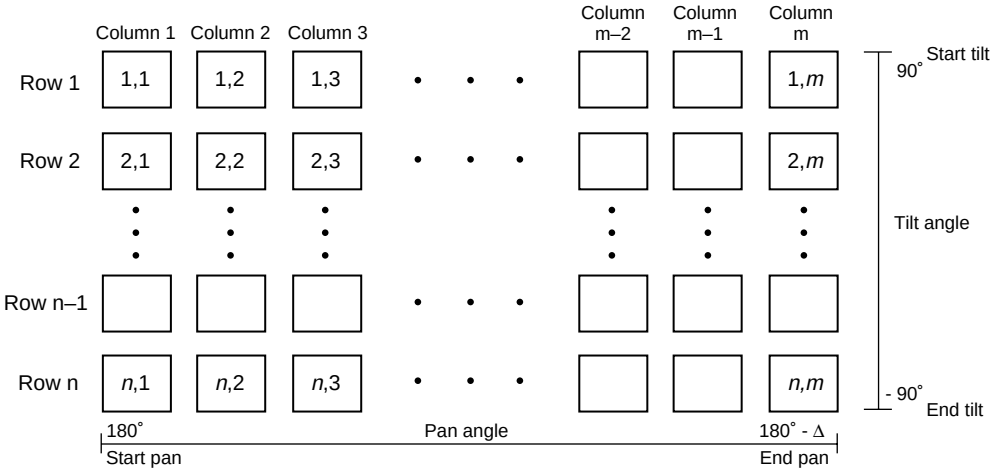
The views that constitute an object node are stored sequentially, as a series of frames in the movie's video track. The authoring tools documentation currently recommends that the first frame be captured with a pan angle of 180 degrees and a tilt angle of 90 degrees. Subsequent frames at that tilt angle should be captured with a +10-degree increment in the pan angle. This scheme gives 36 frames at the starting tilt angle. Then the tilt angle is reduced 10 degrees and the panning process is repeated, resulting in another 36 frames. The tilt angle is gradually reduced until 36 frames are captured at tilt angle –90 degrees. In all, this process results in 684 (that is, 19×36) separate frames.

IMPORTANT

The number of frames captured, the starting and ending pan and tilt angles, and the increments between frames are completely under the control of the author of a QuickTime VR movie. ▲

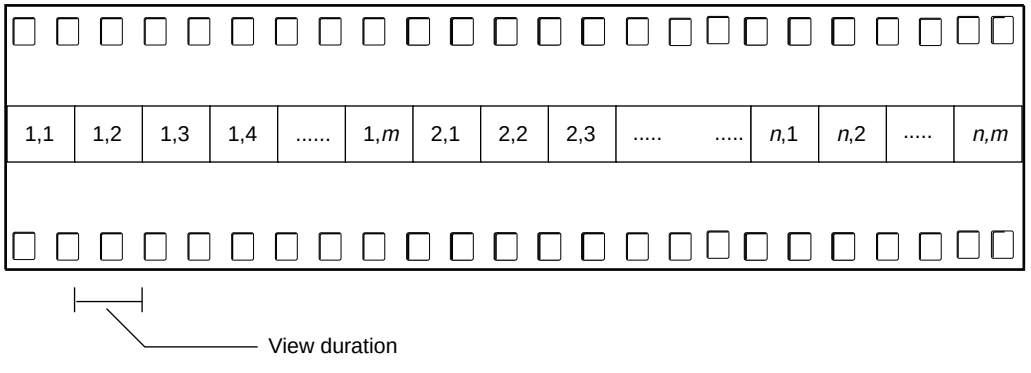
The individual frames of the object can be interpreted as a two-dimensional **object image array** (or **view array**), shown in Figure 1-4. For a simple object (that is, an object with no frame animation or alternate view states), the upper-left frame is the first captured image. A row of images contains the images captured at a particular tilt angle; a column of images contains the images captured at a particular pan angle. Accordingly, turning an object one step to the left is the same as moving one cell to the right in the image array, and turning an object one step down is the same as moving one cell down in the image array. As you'll see later, you can programmatically set the current view of an object either to a specific pan and tilt angle or to a view specified by its row and column in the object image array.

Figure 1-4 An object image array



In the movie file, the image array is stored as a one-dimensional sequence of frames in the movie's video track, as illustrated in Figure 1-5.

Figure 1-5 An object image track



Note
QuickTime VR object nodes were originally designed as a means of showing a 3D object from different pan and tilt angles. However, there is no restriction on the content of the frames stored in an object image array. In other words, the individual frames do not have to be views of the same object from different pan and tilt angles. Some clever movie authors have used this fact to develop intriguing object nodes that are not simply movies of rotating objects. In these cases, the use of pan and tilt angles to specify a view is less meaningful than the use of row and column numbers. Nonetheless, you can always use either pan and tilt angles or row and column numbers to select a view. ◆

Each view of an object occupies the same amount of time in the object node's video track. This amount of time (the **view duration**) is arbitrary, but it is stored in the movie file. When a view is associated with only one frame, the QuickTime VR movie controller displays that frame by changing the current time of the movie to the start time of that view.

It's possible, however, to have more than one frame in a particular object view. Moreover, the number of frames per view can be different from view to view. The only restriction imposed by QuickTime VR is that the view duration be constant throughout all views in a single object node.

Having multiple frames per view is useful in several cases. First, you might want to display one frame if the mouse button is up but a different frame if the mouse button is down. To support this, QuickTime VR allows the VR movie author to include more than one **view state** in an object movie. A view state is an alternate set of images that are displayed, depending on the state of the mouse button.

Note
Alternate view states are stored as separate object image arrays that immediately follow the preceding view state in the object image track. Each state does not need to contain the same number of frames. However, the total movie time of each view state in an object node must be the same. ◆

Another reason to have multiple frames in a particular object view is to display a **frame animation** when that view is the current view. When frame animation is enabled, the QuickTime VR movie controller plays all frames, in sequence, in the current view. You could use frame animation, for instance, to display a

flickering flame on a candle. The rate at which the frames are displayed depends on the view duration and the **frame rate** of the movie (which is stored in the movie file but can be changed programmatically). If the current play rate is nonzero, then the movie controller plays all frames in the view duration. If the current view has multiple states, then the movie controller plays all frames in the current state (which can be set programmatically).

Note
The frames in a frame animation are stored sequentially in each animated view of the object. Each view does not need to contain the same number of frames (so that a view that is not animated can contain only one frame). However, the view duration of each view in an object node must be the same. In some cases, it is best to duplicate the scene frame to get the same view durations and let the compressor remove the extra data. See the chapter "QuickTime VR Atom Containers" for complete information on how object nodes are stored in QuickTime VR movies. ◆

An object movie can be set to play, in order, all the views in the current row of the object image array. This is **view animation.** For both view and frame animation, an object node has a set of **animation settings** that specify characteristics of the movie while it is playing. For example, if a movie's animate view frames flag is set and there are different frames in the current view duration, the movie controller plays an animation at the current view of the object. That is, the movie controller displays all frames in the appropriate portion of the view duration and, if the `kQTVRWrapPan` control setting is on, it starts over when it reaches the segment boundary. If the animate view frames flag is not set, the movie controller stops displaying frames when it reaches the segment boundary. See "Animation Settings" (page 72) for a complete description of the available animation settings.

Panorama Nodes

The data used to represent a panorama is stored as a single **panoramic image** that contains the entire panorama. The movie author creates this image by stitching together individual overlapping digitized photographs of the scene (or by using a 3D renderer to generate an artificial scene). Currently, these images are cylindrical projections of the panorama. Viewed by itself, the panoramic image appears distorted, but it is automatically corrected at runtime when it is

displayed by the QuickTime VR movie controller. Figure 1-6 shows a panoramic image.

Figure 1-6 The panoramic image used to generate panoramic views

A **panorama view** is completely described by its node ID, field of view, pan angle, and tilt angle. As with object nodes, a panoramic node's pan angle can range from 0 degrees to 360 degrees. Increasing the pan angle has the effect of turning one's view to the left. When the user is looking directly into the horizon, the tilt angle is 0. Increasing the tilt angle tilts one's view up, while decreasing the tilt angle tilts one's view down.

For a panorama, the pan and tilt angle correspond to a specific point in the panoramic image. When these angles are set, the corresponding point in the panoramic image is displayed in the center of the current viewing rectangle.

IMPORTANT

The current image-warping technology for panoramic nodes, using cylindrical projection, does not allow looking straight up or straight down. Future versions of QuickTime VR, however, will provide other methods of projection that do support looking straight up and straight down. ▲

While a panorama is being displayed, it can be either at rest (static) or in motion. A panorama is in motion when being panned, tilted, or zoomed. A panorama is also in motion when a **transition** (that is, a movement between two items in a movie, such as from one view in a node to another view in the same node, or from one node to another) is occurring. At all other times, the panorama is static. You can change the imaging properties of a panorama to control the quality and speed of display during rest or motion states. By default,

QuickTime VR sacrifices quality for speed during motion but displays at highest quality when at rest.

When a transition is occurring, you can specify that a special visual effect, called a **transition effect,** be displayed. The only transitional effect currently supported is a **swing transition** between two views in the same node. When the swing transition is enabled and a new pan angle, tilt angle, or field of view is set, the movie controller performs a smooth swing to the new view (rather than a simple jump to the new view). In the future, other transitional effects may be supported.

With version 2.1, QuickDraw VR is capable of using tweening control data that affects the pan angle, tilt angle, and field of view. For information about tweening, please see the *QuickTime 3.0 Reference*.

Hot Spots

Both panoramic nodes and object nodes support arbitrarily shaped **hot spots,** regions in the movie image that permit user interaction. When the cursor is moved over a hot spot (and perhaps when the mouse button is also clicked), QuickTime VR changes the cursor as appropriate and performs certain actions. Which actions are performed depends on the type of the hot spot. For instance, clicking a link hot spot moves the user from one node in a scene to another.

Hot spots can be either enabled or disabled. When a hot spot is enabled, QuickTime VR changes the cursor as it moves in and out of hot spots and responds to mouse button clicks and other user actions. Your application can install callback procedures to respond to mouse actions. When a hot spot is disabled, however, it effectively doesn't exist as far as the user is concerned: QuickTime VR does not change the cursor or execute your callback procedures.

The QuickTime VR Manager provides a number of functions that you can use to manage hot spots. See "Managing Hot Spots" (page 105), for details.

Viewing Limits and Constraints

The data in a panoramic image and in an object image array imposes a set of viewing restrictions on the associated node. For example, a particular panoramic node might be a **partial panorama** (a panorama that is less than 360 degrees). Similarly, the object image array for a particular object node might include views for tilt angles only in a restricted range, say, +45 degrees to –45 degrees (instead of the more usual +90 degrees to –90 degrees). The

allowable ranges of pan angles, tilt angles, and fields of view are the **viewing limits** for the node. Viewing limits are determined at the time a node is authored and are imposed by the data stored in the movie file.

It's possible to impose additional viewing restrictions at runtime. For instance, a game developer might want to limit the amount of a panorama visible to the user until the user achieves some goal (such as touching all the visible hot spots in the node). These additional restrictions are the **viewing constraints** for the node. As you might expect, a viewing constraint must always lie in the range established by the node's viewing limits. By default (that is, if the movie file doesn't contain any viewing constraint atoms, and no constraints have been imposed at runtime), a node's viewing constraints coincide with its viewing limits.

Each node also has a set of **control settings,** which determine the behavior of the QuickTime VR movie controller when the user reaches a viewing constraint. For example, the `kQTVRWrapPan` control setting determines whether the user can wrap around from the current pan constraint maximum value to the pan constraint minimum value (or vice versa) using the mouse or arrow keys. When this setting is enabled, panning past the maximum or minimum pan constraint is allowed. When this setting is disabled, the user cannot pan across the current viewing constraints; when the user reaches a viewing constraint, further panning in that direction is disabled.

Displaying Files While Downloading

Support for viewing QuickTime VR movie files that are being downloaded from the Internet has steadily improved with each new version of QuickTime and QuickTime VR. How a VR movie appears and behaves as it is being downloaded with QuickTime 3 and QuickTime VR 2.1 depends on how and when the VR movie itself was authored.

For older QuickTime VR movies authored with QuickTime 2.0, the user must wait for the entire movie to download before seeing the movie. While the movie is downloading, the QuickTime placeholder picture is visible.

For panoramas authored with QuickTime 2.5 or later, but not specially authored for the Web, the user sees a background grid appear and panorama tiles show up as they are downloaded.

For objects, the first frame of the object appears (usually the top of the object), and subsequent frames become available as they are downloaded. The user can

manipulate the movie as normal while the movie is being downloaded. Hot spots are not active while the movie is being downloaded.

When downloading QuickTime VR panoramas that have been specially authored for the Web, a preview image of the entire panorama appears after about ten per cent of the movie has been downloaded. The high resolution tiles corresponding to the default view appear next, followed by the rest of the tiles. When downloading object movies that are authored for the Web, the default view frame appears first. The remaining frames that make up the row for the default view are downloaded next, followed by frames for any other rows in the object. The user can manipulate the movie as normal while the movie is being downloaded. Hot spots are not active while the movie is being downloaded.

See “Optimizing QuickTime VR Movies for Web Playback” (page 286) in Chapter 5, “Creating QuickTime VR Movies.”

CHAPTER 2

QuickTime VR Manager

Contents

This chapter describes the QuickTime VR Manager, the part of QuickTime 3 that your application can use to interact with QuickTime VR. QuickTime VR is an imaging technology that allows users to explore and examine photorealistic, three-dimensional virtual worlds using standard interaction devices, such as the mouse and keyboard. You can use the QuickTime VR Manager—in conjunction with QuickTime—to open and display QuickTime VR objects and panoramas, change the viewing angle or zoom level, handle mouse events for QuickTime VR movies, and perform other operations on these movies.

To use this chapter, you should already be familiar with QuickTime, as described in *Inside Macintosh: QuickTime*. You need to know how to open and display QuickTime movies, because QuickTime VR objects and panoramas are stored as QuickTime movie tracks. If you need direct access to the movie data stored in an atom container, you also need to be familiar with the atom routines introduced in QuickTime version 2.1. See *QuickTime 3 Reference* for information about the atom routines. See the chapter "QuickTime VR Atom Containers" in this book for a description of the atom containers in a QuickTime VR movie file.

About the QuickTime VR Manager

The **QuickTime VR Manager** is the part of QuickTime 3 that provides an application programming interface for controlling QuickTime VR objects and panoramas. You can use the QuickTime VR Manager to

- display panoramas and objects
- perform basic orientation, positioning, and animation control
- intercept and override QuickTime VR's mouse tracking
- modify the display quality
- intercept and override QuickTime VR's default hot spot behavior
- composite flat or perspective overlays (such as QuickDraw 3D objects or QuickTime movies)
- specify transition effects
- get the viewing limits of a node and get and set a node's viewing constraints
- control QuickTime VR's memory usage

- intercept calls to some QuickTime VR Manager functions and modify their behavior

This section describes the main concepts with which you need to be familiar to use the QuickTime VR Manager. See "Using the QuickTime VR Manager" (page 42) for code examples showing how to use the QuickTime VR Manager.

IMPORTANT

You do not need to use the QuickTime VR Manager simply to open and display a QuickTime VR movie. The QuickTime VR movie controller automatically provides the basic mouse-and-keyboard-driven interface and handles all necessary memory allocation. You need to use the QuickTime VR Manager only if you want to exercise programmatic control over object or panoramic nodes. ▲

QuickTime VR Movie Instances

Almost all the QuickTime VR Manager's functions operate on a QuickTime VR movie instance (defined by the `QTVRInstance` data type). A **QuickTime VR movie instance** is an identifier for a particular QuickTime VR movie. You obtain a QuickTime VR movie instance by calling the `QTVRGetQTVRInstance` function. (See "Creating QuickTime VR Movie Instances" (page 44) for an example.)

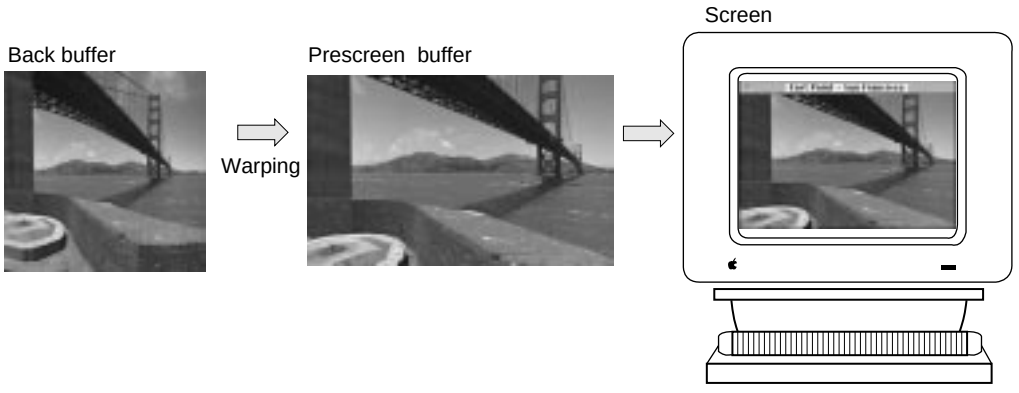
IMPORTANT

There is no need to dispose of a movie instance that you've obtained by calling `QTVRGetQTVRInstance`. ▲

Buffers

For panoramic nodes, QuickTime VR maintains several buffers that it uses to hold the panoramic image before and after the warping that is applied to correct the cylindrical distortion of the original panoramic image. All or part of the uncorrected panoramic image is stored in QuickTime VR's **back buffer.** The corrected image for a particular view (that is, for a particular pan angle, tilt angle, and field of view) is stored in another buffer, the **prescreen buffer** (or **front buffer**). During screen updates, the contents of the prescreen buffer are copied into the graphics world associated with the panoramic node. Figure 2-1 illustrates the internal buffers maintained by QuickTime VR.

Figure 2-1 QuickTime VR's internal buffers



The QuickTime VR Manager allows applications limited access to the contents of the back and prescreen buffers. You can draw directly into the back buffer by installing a **back buffer imaging procedure,** which is called at preestablished times (for instance, whenever an update event occurs for the window containing the movie). You can also draw directly into the prescreen buffer by installing a **prescreen buffer imaging completion procedure,** which is called each time QuickTime VR is finished drawing an image into the prescreen buffer. You can use a prescreen buffer imaging completion procedure to add graphical elements to an image before it is copied to the screen.

Memory Management

QuickTime VR can require large amounts of memory to store its internal representation of the uncorrected image associated with a panoramic node, which is stored in the back buffer. To provide flexibility when operating with limited amounts of memory, a movie's author can include several different resolutions of an image, in different video tracks in the movie file. By default, QuickTime VR selects the highest resolution image available. When memory is limited, however, QuickTime VR selects the image with the highest resolution that fits into the memory it can allocate for its back buffer.

The QuickTime VR Manager provides functions that you can use to determine what resolutions are available and to get and set the current resolution of a panoramic node. You can also use QuickTime VR Manager functions to

override the default behavior for loading data into the back buffer. By default, if enough memory is available, QuickTime VR allocates a back buffer that is large enough to hold the entire uncorrected panoramic image.

Using the QuickTime VR Manager

This section illustrates basic ways of using the QuickTime VR Manager. In particular, it provides source code examples that show how you can

- determine whether the QuickTime VR Manager is available in the current operating environment
- initialize the QuickTime VR Manager
- display a QuickTime VR movie in a window
- create QuickTime VR movie instances
- manipulate a node's pan and tilt angles
- zoom in and out
- install an intercept procedure
- define node-entering and node-leaving procedures
- manage QuickTime VR's panoramic image buffers

Note
The code examples shown in this section provide only rudimentary error handling. ◆

Determining That the QuickTime VR Manager Is Available

Before calling any QuickTime VR Manager routines, you need to verify that the QuickTime VR Manager is available in the current operating environment and that it has the capabilities you need. For the Mac OS, you can verify that the QuickTime VR Manager is available by calling the `Gestalt` function with the `gestaltQTVRMgrAttr` selector. `Gestalt` returns, in its second parameter, a long word whose value encodes the attributes of the QuickTime VR Manager. Listing 2-1 illustrates how to determine whether the QuickTime VR Manager is available.

Listing 2-1 Checking for the availability of the QuickTime VR Manager

```
Boolean MyHasQTVRManager (void)
{
    OSErr           myErr;
    long            myAttrs;
    Boolean         myHasQTVRMgr = false;

    myErr = Gestalt(gestaltQTVRMgrAttr, &myAttrs);
    if (myErr == noErr)
        if (myAttrs & (1 << gestaltQTVRMgrPresent))
            myHasQTVRMgr = true;

    return myHasQTVRMgr;
}
```

You can also use the `Gestalt` function to get information about other attributes of the QuickTime VR Manager. See "Gestalt Selector and Response Values" (page 57) for details on the QuickTime VR Manager attributes you can query.

Note
The `Gestalt` function is available with all operating systems. On those systems that require a call to `InitializeQTML`, `Gestalt` is available after calling `InitializeQTML`. On those systems, calling `InitializeQTVR` is still required after calling `Gestalt`, and the value returned from `InitializeQTVR` must be checked even when the call to `Gestalt` is successful, so the call to `Gestalt` is not necessary, but it can be useful in determining the version and features of the QuickTime VR software that is installed. ◆

Initializing the QuickTime VR Manager

In a Windows environment, before your application can call any QuickTime VR Manager routines, you have to call `InitializeQTVR` so that QuickTime VR can set up its internal data structures. If you make any other calls to the QuickTime VR Manager before calling `InitializeQTVR`, those calls return either a numerical value of zero or an error code of –30555 (`qtvrUninitialized`), which indicates that QuickTime VR has not been initialized. Similarly, functions with

Boolean return types return `false` and functions with OSType return types return `'????'`.

When your application or process has finished using QuickTime VR, it should call `TerminateQTVR`.

You can call `InitializeQTVR` and `TerminateQTVR` more than once; they are reference-counted and nestable.

Note
The `InitializeQTVR` and `TerminateQTVR` routines are required for QuickTime VR to run in a Windows environment. They neither compile nor link in the Mac OS environment.

Creating QuickTime VR Movie Instances

As mentioned earlier, most QuickTime VR Manager functions operate on a QuickTime VR movie instance (defined by the `QTVRInstance` data type), which identifies a particular QuickTime VR movie. You can get a QuickTime VR movie instance by calling the `QTVRGetQTVRInstance` function, as illustrated in Listing 2-2.

Listing 2-2 Getting a QuickTime VR movie instance

```
QTVRInstance MyGetQTVRInstanceFromMC (MovieController theController)
{
    Track               myTrack = nil;
    QTVRInstance        myInstance = nil;
    Movie               myMovie;

    //Get the movie from the movie controller.
    myMovie = MCGetMovie(theController);

    if (myMovie) {
        //Get the first QTVR track in the movie.
        myTrack = QTVRGetQTVRTrack(myMovie, 1);

        //Get a QTVR instance for that QTVR track.
        if (myTrack) {
```

```
            QTVRGetQTVRInstance(myInstance, myTrack, theController);
            //Set our units to be degrees.
            if (myInstance)
                QTVRSetAngularUnits(myInstance, kQTVRDegrees);
        }
    }

    return(myInstance);
}
```

To get a QuickTime VR movie instance, you first need to obtain a **QTVR track,** a special type of QuickTime track that maintains a list of the nodes in the scene. A single QuickTime movie file can contain more than one QuickTime VR scene and hence more than one QTVR track, so you need to specify which QTVR track you want by calling the `QTVRGetQTVRTrack` function with the index of the desired track. Listing 2-2 simply gets the first QTVR track in the specified movie.

Note
Movies made with QuickTime VR 1.0 do not contain a QTVR track. When you call `QTVRGetQTVRTrack` with such a movie, the function returns the appropriate QuickTime track. ◆

After getting the desired QTVR track, the `MyGetQTVRInstanceFromMC` function defined in Listing 2-2 calls the `QTVRGetQTVRInstance` function to obtain a QuickTime VR movie instance. Finally, `MyGetQTVRInstanceFromMC` calls `QTVRSetAngularUnits` to ensure that all angles passed to QuickTime VR functions are interpreted as degrees.

Note
A QuickTime VR movie instance is essentially a pointer to a data structure maintained privately by QuickTime VR. You obtain a movie instance by calling `QTVRGetQTVRInstance`, but you do not need to dispose of that instance. A QuickTime VR movie instance remains valid until you dispose of the QuickTime movie controller (by calling `DisposeMovieController`). ◆

Manipulating Viewing Angles and Zooming

Perhaps the simplest use of the QuickTime VR Manager is to manipulate the current viewing characteristics of an object or panoramic node. You can use the `QTVRGetPanAngle` and `QTVRSetPanAngle` functions to manipulate the pan angle, and you can use the `QTVRGetTiltAngle` and `QTVRSetTiltAngle` functions to manipulate the tilt angle. Listing 2-3 illustrates how to pan or tilt a specific number of degrees in a specific direction.

Listing 2-3 Changing the viewing angle

```
#define kDirLeft    0L
#define kDirRight   1L
#define kDirUp      2L
#define kDirDown    3L

Boolean MyGoDirByDegrees (QTVRInstance theInstance, long theDir, float theAmt)
{
    float       theAngle;
    Boolean     theMoved = false;   //Did calling this routine result in a movement?

    switch (theDir) {
        case kDirUp:
            theAngle = QTVRGetTiltAngle(theInstance);
            QTVRSetTiltAngle(theInstance, theAngle + theAmt);
            break;
        case kDirDown:
            theAngle = QTVRGetTiltAngle(theInstance);
            QTVRSetTiltAngle(theInstance, theAngle - theAmt);
            break;
        case kDirLeft:
            theAngle = QTVRGetPanAngle(theInstance);
            QTVRSetPanAngle(theInstance, theAngle + theAmt);
            break;
        case kDirRight:
            theAngle = QTVRGetPanAngle(theInstance);
            QTVRSetPanAngle(theInstance, theAngle - theAmt);
            break;
        default:
            break;
```

```
    }

    //Now update the image on the screen.
    QTVRUpdate(theInstance, kQTVRStatic);

    //Determine whether a movement actually occurred.
    switch (theDir) {
        case kDirUp:
        case kDirDown:
            theMoved = (theAngle != QTVRGetTiltAngle(theInstance));
            break;
        case kDirLeft:
        case kDirRight:
            theMoved = (theAngle != QTVRGetPanAngle(theInstance));
            break;
        default:
            break;
    }

    return(theMoved);
}
```

`MyGoDirByDegrees` is relatively simple. It first determines the direction in which to move, gets the current pan or tilt angle, and then sets a new pan or tilt angle by adding or subtracting the desired displacement to that angle. Notice that `MyGoDirByDegrees` calls the `QTVRUpdate` function to update the image on the screen. This update is necessary whenever you change a viewing characteristic programmatically.

Once the new viewing angle has been set and the new image has been displayed, the `MyGoDirByDegrees` function determines whether the new pan or tilt angle differs from the pan or tilt angle on entry and passes back a Boolean value to indicate whether the call to `MyGoDirByDegrees` changed the pan or tilt angle. (The new angle may not be different because, for example, the value was already at some limit or constraint. This information might be useful for determining whether to enable or disable some visual effect in the scene.)

Zooming in or out is just as simple as panning or tilting. For both objects and panoramas, you zoom in or out by changing the field of view of the node. Listing 2-4 defines a function that zooms in or out by a predetermined amount.

Listing 2-4 Changing the field of view

```
#define kDirIn      4L
#define kDirOut     5L

void MyZoomInOrOut (QTVRInstance theInstance, long theDir)
{
    float   theFloat;

    theFloat = QTVRGetFieldOfView(theInstance);
    switch (theDir) {
        case kDirIn:
            theFloat = theFloat / 2.0;
            break;
        case kDirOut:
            theFloat = theFloat * 2.0;
            break;
        default:
            break;
    }
    QTVRSetFieldOfView(theInstance, theFloat);
    QTVRUpdate(theInstance, kQTVRStatic);
}
```

The `MyZoomInOrOut` function defined in Listing 2-4 simply doubles or halves the current field of view, depending on whether you're zooming out or in.

Intercepting QuickTime VR Manager Routines

The QuickTime VR Manager provides support for intercepting some of its routines. To intercept a routine, you need to define and install an **intercept procedure,** a function that is executed in addition to (or instead of) the QuickTime VR Manager function it's intercepting.

Typically, you'll use an intercept procedure to augment the behavior of a QuickTime VR Manager function. For instance, you might intercept the `QTVRSetPanAngle` function to play a specific sound when the user moves to a particular pan angle. In this case, you would have the QuickTime VR Manager execute the `QTVRSetPanAngle` function, and then you would play the appropriate sound.

Alternatively, you might want to override the intercepted function altogether. For instance, you might intercept the `QTVRTriggerHotSpot` function so that when the user clicks a custom hot spot, you can respond accordingly. In this case, there is no need to have the QuickTime VR Manager execute the `QTVRTriggerHotSpot` function.

You declare an intercept procedure like this:

```
pascal void MyInterceptProc (
                    QTVRInstance qtvr,
                    QTVRInterceptPtr qtvrMsg,
                    SInt32 refCon,
                    Boolean *cancel);
```

The `qtvr` parameter is the instance with which you're concerned. The `qtvrMsg` parameter is a pointer to an **intercept structure,** which contains information about the routine being intercepted and its parameters. The `refCon` parameter is a long integer available for use by your application. Finally, your intercept procedure should set the `cancel` parameter to indicate whether the QuickTime VR Manager should execute the intercepted function when your intercept procedure has finished (`false`) or should not execute the function (`true`).

Note
If you don't set the `cancel` parameter before exiting your intercept procedure, its value is by default set to `false` (indicating that the intercepted function should be executed). ◆

The intercept structure is defined by the `QTVRInterceptRecord` data type:

```
typedef struct QTVRInterceptRecord {
    SInt32                              reserved1;
    SInt32                              selector;
    SInt32                              reserved2;
    SInt32                              reserved3;
    SInt32                              paramCount;
    void                                *parameter[6];
} QTVRInterceptRecord, *QTVRInterceptPtr;
```

As you can see, many of the fields of an intercept structure are reserved. The interesting fields are `selector`, `paramCount`, and `parameter`. The `selector` field is

an **intercept selector,** a constant that indicates which routine has triggered your intercept procedure. You can, if you wish, install a single intercept procedure for all intercepted functions. In that case, you can inspect the `selector` field of the intercept structure passed to your intercept routine to determine how to respond. The QuickTime VR Manager currently defines these intercept selectors:

```
typedef enum QTVRProcSelector {
    kQTVRSetPanAngleSelector                    = 0x2000,
    kQTVRSetTiltAngleSelector                   = 0x2001,
    kQTVRSetFieldOfViewSelector                 = 0x2002,
    kQTVRSetViewCenterSelector                  = 0x2003,
    kQTVRMouseEnterSelector                     = 0x2004,
    kQTVRMouseWithinSelector                    = 0x2005,
    kQTVRMouseLeaveSelector                     = 0x2006,
    kQTVRMouseDownSelector                      = 0x2007,
    kQTVRMouseStillDownSelector                 = 0x2008,
    kQTVRMouseUpSelector                        = 0x2009,
    kQTVRTriggerHotSpotSelector                 = 0x200A,
    kQTVRGetHotSpotTypeSelector                 = 0x200B
} QTVRProcSelector;
```

The `parameter` field of the intercept structure is an array that holds, in order, the parameters that were passed to the intercepted function, minus the QTVR instance parameter. For example, if you intercept the `QTVRSetPanAngle` function, the `parameter` array contains a single member, a pointer to a floating-point value that is the new pan angle. You can determine how many members the `parameter` array contains by inspecting the `paramCount` field of the intercept structure.

Listing 2-5 defines a simple intercept procedure that is called whenever the QuickTime VR Manager function `QTVRSetPanAngle` is called. The intercept procedure calls an application-defined function, `MyPlayPanSound`, to play a sound for the new pan angle.

Listing 2-5 Intercepting the `QTVRSetPanAngle` function (version 1)

```
#define MyPi (3.1415926535898)
#define RadiansToDegrees(x) ((x) * 180.0 / MyPi)

pascal void MyInterceptProc (QTVRInstance theInstance,
                QTVRInterceptPtr theMsg, SInt32 refcon, Boolean *cancel)
{
```

```
    Boolean     cancelInterceptedProc = false;
    float       theAngle, *theAnglePtr;

    switch (theMsg->selector) {
        case kQTVRSetPanAngleSelector:
            theAnglePtr = theMsg->parameter[0];
            theAngle = *theAnglePtr;
            theAngle = RadiansToDegrees(theAngle);
            MyPlayPanSound(theAngle);       //Play a sound for the angle.
            break;

        default:
            break;
    }

    *cancel = cancelInterceptedProc;
}
```

IMPORTANT

Angular values in the `parameter` field of an intercept structure are always returned in radians, regardless of the current angular unit. In addition, a floating-point value is always passed as a pointer to a floating-point value. ▲

The intercept procedure defined in Listing 2-5 returns the value `false` in the `cancel` parameter. This indicates that the QuickTime VR Manager should call the intercepted function after the intercept procedure exits. If the `cancel` parameter is set to `true`, the QuickTime VR Manager does not call the intercepted function. This is useful if you want to replace the intercepted function altogether or if you want to call the intercepted function from within your intercept procedure. For example, if you want to play a sound *after* the new pan angle is displayed, you can define an intercept procedure like the one in Listing 2-6.

Listing 2-6 Intercepting the `QTVRSetPanAngle` function (version 2)

```
pascal void MyInterceptProc (QTVRInstance theInstance,
    QTVRInterceptPtr theMsg, SInt32 refcon, Boolean *cancel)
{
    Boolean cancelInterceptedProc = false;
```

```
    switch (theMsg->selector) {
        case kQTVRGetHotSpotTypeSelector:
        {
        OSType hsType;
        QTVRCallInterceptedProc (theInstance, theMsg);
        hsType = * ((UInt32 *) theMsg->parameter[1]);
        // Turn all url hotspots into undefined hotspots
        if (hsType == kQTVRHotSpotURLType)
            * ((UInt32 *) theMsg->parameter[1]) = kQTVRHotSpotUndefinedType;
        cancelInterceptedProc = true;
        break;
        }

    default:
        break;
}

*cancel = cancelInterceptedProc;
}
```

The intercept procedure defined in Listing 2-6 looks at the hot spot type returned by the call to `QTVRCallInterceptedProc` and changes it to undefined if it is a URL hot spot.

Notice that the new intercept procedure returns the value `true` in the `cancel` parameter, indicating that the QuickTime VR Manager should *not* call the intercepted function after the intercept procedure returns. If the intercept procedure returns `false`, then the intercepted function will be called twice (once because you call `QTVRCallInterceptedProc` and a second time because you return `false` in the `cancel` parameter).

IMPORTANT

You should use the `QTVRCallInterceptedProc` function only in an intercept procedure. Moreover, you should use `QTVRCallInterceptedProc` instead of the function you're intercepting. If you called `QTVRSetPanAngle` directly in Listing 2-6, your intercept procedure would be called repeatedly until your stack overflowed. ▲

You install an intercept procedure by calling the `QTVRInstallInterceptProc` function, as shown in Listing 2-7.

Listing 2-7 Installing an intercept procedure

```
QTVRInterceptUPP MyInstallInterceptProcedure (QTVRInstance theInstance)
{
    QTVRInterceptUPP    theInterceptProc;

    theInterceptProc = NewQTVRInterceptProc(MyInterceptProc);
    QTVRInstallInterceptProc(theInstance, kQTVRSetPanAngleSelector,
                                            theInterceptProc, 0, 0);
    return  theInterceptProc
}
. . .
myProc = MyInstallInterceptProcedure(qtvr);
```

`QTVRInstallInterceptProc` takes an intercept selector to determine which QuickTime VR Manager function to intercept. If you wish, you can define a single intercept procedure and use the intercept selector passed to it in the `selector` field of `theMsg` to decide how to respond.

When you no longer need the intercept procedure you should call `QTVRInstallInterceptProc` again with the same selector and a `nil` procedure pointer and then call `DisposeRoutineDescriptor` on `myProc`.

Entering and Leaving Nodes

The QuickTime VR Manager provides a way for you to be notified whenever the user is about to enter a new node or leave the current node. You can then react to these notifications in whatever manner you choose. For example, when the user is about to enter a new node, you might determine the name of that new node and display the name or other information about the node. Similarly, when the user is about to leave the current node, you might initiate a custom node-to-node transition effect. Alternatively, you can cancel the move to the other node; this might be useful in a game when the user hasn't yet searched the current node completely or accomplished some other predefined task in that node.

To be informed that the user is about to enter a new node, you define and install a **node-entering procedure.** Listing 2-8 illustrates a simple node-entering procedure that determines the name of the new node and then utters that name.

Listing 2-8 Informing the user of a new node's name

```
pascal OSErr MyEnteringNodeProc (QTVRInstance theInstance,
                                    UInt32 theNodeID, SInt32 refCon)
{
    Str255          theString;
    OSErr           theErr;

    theErr = MyGetNodeName(theInstance, theNodeID, &theString);
    if (!theErr)
        SpeakString(theString);

    return(theErr);
}
```

Note
See Listing 4-1 (page 256) for the definition of the function `MyGetNodeName` defined in Listing 2-8. ◆

You install a node-entering procedure by calling the `QTVRSetEnteringNodeProc` function, like this:

```
theErr = QTVRSetEnteringNodeProc(theInstance,
                    NewQTVREnteringNodeProc(MyEnteringNodeProc), 0, 0);
```

To be informed that the user is about to leave the current node, you define and install a **node-leaving procedure.** Listing 2-9 illustrates a simple node-leaving procedure that prevents the user from leaving the current node unless all hot spots in the node have been triggered.

Listing 2-9 Leaving a node

```
pascal OSErr MyLeavingNodeProc (QTVRInstance theInstance,
                                UInt32 fromNodeID, UInt32 toNodeID,
                                Boolean *cancel, MyDataPtr theDataPtr)
{
    Boolean theUserCanLeave = false;    //By default, user can't leave.

    if (theDataPtr->allHotSpotsTouched)
        theUserCanLeave = true;
```

```
    *cancel = !theUserCanLeave;
    return(noErr);
}
```

Before returning, your node-leaving procedure should set the Boolean value pointed to by the `cancel` parameter to `false` to accept the move from `fromNodeID` to `toNodeID`. Set that value to `true` to cancel the move and remain at the node specified by the `fromNodeID` parameter. The procedure defined in Listing 2-9 simply reads some private data to determine whether to allow the user to leave the current node.

You install a node-leaving procedure by calling the `QTVRSetLeavingNodeProc` function, like this:

```
theErr = QTVRSetLeavingNodeProc(theInstance,
        NewQTVRLeavingNodeProc(MyLeavingNodeProc), (SInt32)&theData, 0);
```

In a multinode movie, your node-entering procedure is not called for the first node. This is because the user is considered to be in the first node as soon as the VR movie is opened, before you have a chance to install your node-entering procedure. If you need to have your node-entering procedure called for the first node, you can execute it explicitly, either before or after you've installed it as a node-entering procedure.

Drawing in the Prescreen Buffer

The QuickTime VR Manager allows you to define a prescreen buffer imaging completion procedure that is called whenever QuickTime VR finishes drawing a panorama image in the prescreen buffer. Typically, your completion procedure adds graphical elements to the image before the buffer is copied to the screen. For instance, a flight simulator could overlay a heads-up display containing information about the aircraft (its altitude, velocity, and so forth).

You install a prescreen buffer imaging completion procedure by passing its address to the `QTVRSetPrescreenImagingCompleteProc` function:

```
theErr = QTVRSetPrescreenImagingCompleteProc(theInstance,
        NewQTVRImagingCompleteProc(MyImagingCompleteProc),
        (SInt32)&theData, 0);
```

Listing 2-10 defines a simple completion routine that overlays a picture onto the screen image.

Listing 2-10 Overlaying images in the prescreen buffer

```
pascal OSErr MyImagingCompleteProc (QTVRInstance, MyDataPtr theDataPtr)
{
    if (theDataPtr->hasLogoPict) {
        GWorldPtr   theOffscreenGWorld;
        GDHandle    theGD;
        Rect        gwRect;
        Rect        picRect;

        // The current graphics world is set to the prescreen buffer.
        GetGWorld (&theOffscreenGWorld, &theGD);
        gwRect = (*(theOffscreenGWorld->portPixMap))->bounds;

        picRect = (*(theDataPtr->logoPict))->picFrame;
        OffsetRect (&picRect, -picRect.left, -picRect.top);
        OffsetRect (&picRect, gwRect.right - (picRect.right + 8),
                gwRect.bottom - (picRect.bottom + 8));
        // Draw logo in lower right corner
        DrawPicture (theDataPtr->logoPict, &picRect);
    }
    return noErr;
}
```

On entry to the prescreen buffer imaging completion routine, the current graphics world is set to QuickTime VR's prescreen buffer. The `MyImagingCompleteProc` function defined in Listing 2-10 retrieves the dimensions of that buffer and then draws a picture in the lower-right corner of that buffer.

QuickTime VR Manager Reference

This section describes the constants, data structures, and routines provided by the QuickTime VR Manager.

Constants

This section describes the constants provided by the QuickTime VR Manager. You use these constants to specify node types, intercepted routines, correction and imaging modes, and other information.

Gestalt Selector and Response Values

You can pass the `gestaltQTVRMgrVers` selector to the `Gestalt` function to get the version of the QuickTime VR Manager. You can pass the `gestaltQTVRMgrAttr` selector to the `Gestalt` function to get information about the QuickTime VR Manager. `Gestalt` returns information to you by setting or clearing bits in the `response` parameter. The selectors and bits currently used are defined by constants:

```
enum {
    gestaltQTVRMgrAttr                  = FOUR_CHAR_CODE('qtvr'),
    gestaltQTVRMgrVers                  = FOUR_CHAR_CODE('qtvv'),
    gestaltQTVRMgrPresent                           = 0,
    gestaltQTVRObjMoviesPresent                     = 1,
    gestaltQTVRCylinderPanosPresent                 = 2
};
```

Constant descriptions

`gestaltQTVRMgrAttr` Return the attributes of the QuickTime VR Manager.

`gestaltQTVRMgrVers` Return the version of the QuickTime VR Manager.

`gestaltQTVRMgrPresent`
This bit is set if the QuickTime VR Manager is present in the current operating environment.

`gestaltQTVRObjMoviesPresent`
This bit is set if the QuickTime VR Manager supports object nodes.

`gestaltQTVRCylinderPanosPresent`
This bit is set if the QuickTime VR Manager supports panoramic nodes that use cylindrical projection.

Note
For complete information about the `Gestalt` function, see the chapter "Gestalt Manager" in *Inside Macintosh: Operating System Utilities.* ◆

Node Types

The `QTVRGetNodeType` function returns a value that specifies the type of a node. Currently, these values can be returned:

```
enum {
    kQTVRPanoramaType               = FOUR_CHAR_CODE('pano'),
    kQTVRObjectType                 = FOUR_CHAR_CODE('obje')
};
```

Constant descriptions

`kQTVRPanoramaType` The node is a panoramic node.

`kQTVRObjectType` The node is an object node.

Node IDs

The `nodeID` parameter to the `QTVRGoToNodeID` function specifies the ID of a node. The QuickTime VR Manager defines these constants for specific nodes:

```
enum {
    kQTVRCurrentNode                                = 0,
    kQTVRPreviousNode                               = 0x80000000,
    kQTVRDefaultNode                                = 0x80000001
};
```

Constant descriptions

`kQTVRCurrentNode` The current node. (Moving to the current node has the effect of restoring the default view in that node.)

`kQTVRPreviousNode` The previous node. The QuickTime VR Manager maintains a list (which is limited only by the available memory) of the nodes visited.

`kQTVRDefaultNode` The default node in the scene.

Angular Unit Types

The `QTVRGetAngularUnits` and `QTVRSetAngularUnits` functions use the following constants to indicate the type of angular unit associated with a QuickTime VR movie instance:

```
typedef enum QTVRAngularUnits {
    kQTVRDegrees                                = 0,
    kQTVRRadians                                = 1
} QTVRAngularUnits;
```

Constant descriptions

`kQTVRDegrees` Angles are specified using degrees. This is the default type of angle specification.

`kQTVRRadians` Angles are specified using radians.

Hot Spot Action Selectors

The `flags` parameter passed to an application-defined mouse over hot spot procedure specifies a type of mouse action for a hot spot. See “Mouse Over Hot Spot Procedure” (page 180). These constants define the available hot spot actions:

```
enum {
    kQTVRHotSpotEnter                           = 0,
    kQTVRHotSpotWithin                          = 1,
    kQTVRHotSpotLeave                           = 2
};
```

Constant descriptions

`kQTVRHotSpotEnter` The cursor has just entered the hot spot.

`kQTVRHotSpotWithin` The cursor is still in the hot spot.

`kQTVRHotSpotLeave` The cursor has just left the hot spot.

Flags Value for Imaging Completion Procedure

The `kQTVRPreScreenEveryIdle` value of the `flags` parameter passed to `QTVRSetPrescreenImagingCompleteProc` causes a drawing attempt on every idle

passed to the movie controller (`MCIdle` or `MCIsPlayerEvent(idle)`). The purpose of the flag is to cause the software to draw as often as possible.

```
enum{
    kQTVRPreScreenEveryIdle                         = 1L << 0
};
```

Intercept Selectors

The QuickTime VR Manager allows you to intercept a number of its functions by installing an intercept procedure. You specify which function you want to intercept by passing an intercept selector to the `QTVRInstallInterceptProc` function. These are the available intercept selectors:

```
typedef enum QTVRProcSelector {
    kQTVRSetPanAngleSelector                        = 0x2000,
    kQTVRSetTiltAngleSelector                       = 0x2001,
    kQTVRSetFieldOfViewSelector                     = 0x2002,
    kQTVRSetViewCenterSelector                      = 0x2003,
    kQTVRMouseEnterSelector                         = 0x2004,
    kQTVRMouseWithinSelector                        = 0x2005,
    kQTVRMouseLeaveSelector                         = 0x2006,
    kQTVRMouseDownSelector                          = 0x2007,
    kQTVRMouseStillDownSelector                     = 0x2008,
    kQTVRMouseUpSelector                            = 0x2009,
    kQTVRTriggerHotSpotSelector                     = 0x200A,
    kQTVRGetHotSpotTypeSelector                     = 0x200B
} QTVRProcSelector;
```

Constant descriptions

`kQTVRSetPanAngleSelector`
: Intercept the `QTVRSetPanAngle` function.

`kQTVRSetTiltAngleSelector`
: Intercept the `QTVRSetTiltAngle` function.

`kQTVRSetFieldOfViewSelector`
: Intercept the `QTVRSetFieldOfView` function.

`kQTVRSetViewCenterSelector`
: Intercept the `QTVRSetViewCenter` function.

`kQTVRMouseEnterSelector`
: Intercept the `QTVRMouseEnter` function.

`kQTVRMouseWithinSelector`
: Intercept the `QTVRMouseWithin` function.

`kQTVRMouseLeaveSelector`
: Intercept the `QTVRMouseLeave` function.

`kQTVRMouseDownSelector`
: Intercept the `QTVRMouseDown` function.

`kQTVRMouseStillDownSelector`
: Intercept the `QTVRMouseStillDown` function.

`kQTVRMouseUpSelector`
: Intercept the `QTVRMouseUp` function.

`kQTVRTriggerHotSpotSelector`
: Intercept the `QTVRTriggerHotSpot` function.

`kQTVRGetHotSpotTypeSelector`
: Intercept the `QTVRGetHotSpotType` function

Constraint Types

The `kind` parameter of the `QTVRGetViewingLimits`, `QTVRGetConstraints`, `QTVRSetConstraints`, and `QTVRWrapAndConstrain` functions specifies the type of angle or other feature of which you want to determine or set the constraints (that is, the limits of the current node). You can pass one of these values in that parameter:

```
enum {
    kQTVRPan                                    = 0,
    kQTVRTilt                                   = 1,
    kQTVRFieldOfView                            = 2,
    kQTVRViewCenterH                            = 4,
    kQTVRViewCenterV                            = 5
};
```

Constant descriptions

`kQTVRPan` The pan angle. The associated value is of type `float`.

`kQTVRTilt` The tilt angle. The associated value is of type `float`.

`kQTVRFieldOfView` The field of view. The associated value is of type `float`.

`kQTVRViewCenterH` The horizontal view center. This value is valid only for the `QTVRWrapAndConstrain` function, which returns the horizontal view center constrained in the current field of view. The minimum and maximum values of the horizontal view center change for every field-of-view setting. If the field of view of the object is the maximum field of view, the object's horizontal view center is constrained to a single value that is the horizontal center of the object image in display coordinates (rounding down any fractional pixels).

`kQTVRViewCenterV` The vertical view center. This value is valid only for the `QTVRWrapAndConstrain` function, which returns the vertical view center constrained in the current field of view. The minimum and maximum values of the vertical view center change for every field-of-view setting. If the field of view of the object is the maximum field of view, the object's vertical view center is constrained to a single value that is the vertical center of the object image in display coordinates (rounding down any fractional pixels).

Correction Modes

The correction mode of a panorama (specified by the `kQTVRImagingCorrection` constant) specifies a type of image correction to be applied when imaging a panoramic view. You can pass one of these values to specify a correction mode:

```
enum {
    kQTVRNoCorrection                       = 0,
    kQTVRPartialCorrection                  = 1,
    kQTVRFullCorrection                     = 2
};
```

Constant descriptions

`kQTVRNoCorrection` Apply no warping. The source panorama is reproduced directly, with no image correction.

`kQTVRPartialCorrection` Apply one-dimensional (horizontal) warping. This kind of correction is often sufficient, especially for outdoor scenes.

`kQTVRFullCorrection`
: Apply two-dimensional warping. This correction mode produces perspectively correct images from the source panorama.

IMPORTANT

Future versions of QuickTime VR may employ warping methods that interpret these correction modes differently. You can always assume, however, that the `kQTVRNoCorrection` mode performs no image correction at all, and that the `kQTVRFullCorrection` mode performs more correction than `kQTVRPartialCorrection` (and hence requires more time to construct the resulting image). ▲

Imaging Modes

The `imagingMode` parameter of the functions `QTVRGetImagingProperty` (page 145), `QTVRSetImagingProperty` (page 146), and several others specifies an imaging mode. These constants define the available imaging modes:

```
typedef enum QTVRImagingMode {
    kQTVRCurrentMode                    = 0,
    kQTVRStatic                         = 1,
    kQTVRMotion                         = 2,
    kQTVRAllModes                       = 100
} QTVRImagingMode;
```

Constant descriptions

`kQTVRCurrentMode`
: The current imaging mode. This value is valid only for the function `QTVRUpdate` (page 148).

`kQTVRStatic`
: The panorama is static.

`kQTVRMotion`
: The panorama is in motion; that is, an action such as panning or zooming is occurring.

`kQTVRAllModes`
: All currently defined imaging modes. You can specify this imaging mode when calling `QTVRSetImagingProperty` to assign a value to a particular imaging property for all imaging modes. You can also use this imaging mode in the `imagingMode` field of a panorama-imaging atom structure to indicate a default value for a particular imaging property

for all imaging modes. For information about panorama-imaging atom structures, see “Panorama-Imaging Atom” (page 246) in Chapter 4, “QuickTime VR Atom Containers.”

Imaging Property Types

The `imagingProperty` parameter of the functions `QTVRGetImagingProperty` (page 145) and `QTVRSetImagingProperty` (page 146) specifies the type of imaging property of a panoramic node whose value you want to get or set. These constants define the available imaging properties:

```
enum {
    kQTVRImagingCorrection                  = 1,
    kQTVRImagingQuality                     = 2,
    kQTVRImagingDirectDraw                  = 3,
    kQTVRImagingCurrentMode                 = 100
};

#define kQTVRImagingDefaultValue            0x80000000
```

Constant descriptions

`kQTVRImagingCorrection`

The correction mode property. This property determines the type of image correction to be applied when imaging a panoramic view. The acceptable values for this property type are described in “Correction Modes” (page 62).

`kQTVRImagingQuality`

The imaging quality property. This property determines the quality of the output image. The acceptable values for this property type are described in “Quality Properties” (page 65).

`kQTVRImagingDirectDraw`

The direct-drawing property. This property determines the immediate destination of an image. The value of this property (as specified by the `propertyValue` parameter) is interpreted as a Boolean value. If the value is `true`, then images are computed and drawn directly to the final destination without first being drawn in the prescreen buffer maintained by QuickTime VR. This value provides

the fastest overall frame rate but may result in relatively slower individual frame drawing for high-quality, high-resolution images on lower performance systems. If the value of this property is `false`, then images are computed and drawn first into the prescreen buffer and then to the final destination. This value provides the shortest imaging time for each individual frame but results in a reduced overall frame rate.

`kQTVRImagingCurrentMode`

The current imaging mode property. When you pass this property selector to the `QTVRGetImagingProperty` function, it returns, in the `propertyValue` parameter, the current imaging mode (that is, either `kQTVRStatic` or `kQTVRMotion`). If the `kQTVRImagingDefaultValue` bit is set, the default imaging mode is returned. You can get but not set the value of this property. You might want to determine the current imaging mode in an imaging callback procedure if what you draw depends on the current imaging mode.

`kQTVRImagingDefaultValue`

The default value specifier. You can OR this value with any of the imaging property types to get or set the default value of that property type.

Note

The `kQTVRImagingDirectDraw` property cannot always be supported. During updates, QuickTime VR's warping engine might ignore a value of `true` and draw the image first into the prescreen buffer and then into the final destination. ◆

Quality Properties

The `propertyValue` parameter of the `QTVRGetImagingProperty` and `QTVRSetImagingProperty` functions specifies the value of an imaging property of a panoramic node. These constants define the available values of the `kQTVRImagingQuality` imaging property:

```
#define codecMinQuality                         0x000L
#define codecLowQuality                         0x100L
#define codecNormalQuality                      0x200L
#define codecHighQuality                        0x300L
#define codecMaxQuality                         0x3FFL
```

The imaging quality setting is interpreted purely as a scalar value, in much the same way as it is for any other QuickTime movie.

The use of `codecMinQuality` always results in the best possible performance without regard to quality, even on systems where a negligible decrease in performance might yield significant improvements in quality. Similarly, the use of `codecMaxQuality` always results in the best possible quality, regardless of performance. Because of the possibility for extreme degradation of performance or quality, the scalar limits should be used only in specialized circumstances. Values of `codecLowQuality` and `codecHighQuality` are recommended for good performance and quality, respectively, across a range of different systems.

Transition Type

The `transitionType` parameter of the function `QTVRSetTransitionProperty` (page 151) specifies a type of transition for a panoramic node. This constant defines the available value of that parameter:

```
enum {
    kQTVRTransitionSwing                        = 1
};
```

Constant description

`kQTVRTransitionSwing`
A transition between two views in a single node.

Transition Properties

The `transitionProperty` parameter of the function `QTVRSetTransitionProperty` (page 151) specifies a type of property for a transition for a panoramic node. These constants define the available values of that parameter:

```
enum {
    kQTVRTransitionSpeed                    = 1,
    kQTVRTransitionDirection                = 2
};
```

Constant descriptions

`kQTVRTransitionSpeed`

The speed at which the transition should occur. The value in the `transitionValue` parameter should be an integer from 1 (the slowest transition speed) through 10 (the fastest transition speed).

`kQTVRTransitionDirection`

The direction in which the transition should occur. The value in the `transitionValue` parameter should be a constant of type `QTVRNudgeControl` that indicates the direction in which the view should swing while moving from the current view to the new view. This direction can be left or right, or it can have the value –1, which causes the default direction to be used. By default, a swing transition occurs along the shortest route between the two views.

Hot Spot Types

QuickTime VR recognizes three hot spot types. Only two of those have an associated info atom: the link hot spot type and the URL hot spot type. There is no specific info atom for the undefined hot spot type, `kQTVRHotSpotUndefinedType` (`'undf'`). However, every undefined hot spot should have a hot spot and hot spot info atom in the node information atom container.

QuickTime VR provides the following constants to specify the type of a hot spot:

```
enum {
    kQTVRHotSpotLinkType                    = 'link',
    kQTVRHotSpotURLType                     = 'url ',
    kQTVRHotSpotUndefinedType               = 'undf'
};
```

Constant descriptions

`kQTVRHotSpotLinkType`
A link hot spot.

`kQTVRHotSpotURLType`
A URL (Universal Resource Locator) hot spot.

`kQTVRHotSpotUndefinedType`
An undefined hot spot type.

IMPORTANT

Apple Computer reserves hot spot types that consist of all lower-case letters. To avoid conflicts with possible future hot spot types defined by Apple Computer, applications that define their own hot spot types should not use all lower-case letters. ▲

Interaction Property Types

The `property` parameter of the `QTVRGetInteractionProperty` and `QTVRSetInteractionProperty` functions specifies a type of user interaction property. For any specific property type, the `value` parameter specifies its current or desired value. These constants define the available interaction property types:

```
enum {
    kQTVRInteractionMouseClickHysteresis            = 1,
    kQTVRInteractionMouseClickTimeout               = 2,
    kQTVRInteractionPanTiltSpeed                    = 3,
    kQTVRInteractionZoomSpeed                       = 4,
    kQTVRInteractionTranslateOnMouseDown            = 101,
    kQTVRInteractionMouseMotionScale                = 102,
    kQTVRInteractionNudgeMode                       = 103
};

#define kQTVRInteractionDefaultValue                0x80000000
```

You can get or set the default value of any of the interaction values by performing a bitwise OR of the property constant with `kQTVRInteractionDefaultValue` and pasting the result into the `property` parameter.

Constant descriptions

`kQTVRInteractionMouseClickHysteresis`

The `value` parameter is interpreted as an unsigned short integer that represents the **mouse-click hysteresis,** the distance, in pixels, from the location of a mouse-down event to the limit within which the cursor is considered not to have moved. In other words, the cursor can move that many pixels during a mouse click and still have the event considered a click. The default mouse-click hysteresis value is 2. This property is valid for panoramas only.

`kQTVRInteractionMouseClickTimeout`

The `value` parameter is interpreted as an unsigned long integer that represents the **mouse-click timeout,** the number of ticks after which a mouse click times out and is automatically switched from a hot spot selection into a pan. The default mouse-click timeout value is 30 ticks (one-half second). This property is valid for panoramas only.

`kQTVRInteractionPanTiltSpeed`

The panning and tilting speed. The `value` parameter is interpreted as an unsigned long integer that represents the relative speed of panning and tilting. This integer should be from 1 (the slowest speed) through 10 (the fastest speed); the default panning and tilting speed is 5. This property is valid only for panoramas.

`kQTVRInteractionZoomSpeed`

The `value` parameter is interpreted as an unsigned long integer that represents the **zooming speed,** the relative speed of zooming in and out. This integer should be from 1 (the slowest speed) through 10 (the fastest speed); the default zooming speed is 5. This property is valid for both objects and panoramas.

`kQTVRInteractionTranslateOnMouseDown`

The translate flag. The `value` parameter is interpreted as a Boolean value that indicates whether translate mode is enabled (`true`) or disabled (`false`). When **translate mode** is enabled, the user can no longer pan or tilt using the mouse; instead, dragging the cursor causes the object to translate. The default translate flag value is `false`. This property is valid for objects only.

`kQTVRInteractionMouseMotionScale`
: The `value` parameter is interpreted as a pointer to a floating-point number that represents the **mouse-motion scale,** the number of degrees or radians that an object or panorama is panned or tilted when the cursor is dragged the entire width of the object bounding box. The default value is 180.0. This property is valid for objects only.

`kQTVRInteractionNudgeMode`
: This parameter lets you set the QuickTime VR nudge mode to either rotate, translate, or be the same as the current mouse mode. See "Nudge Mode" (page 80).

Viewing Constraints

The `constraints` parameter of the `QTVRGetConstraintStatus` function is a pointer to a long integer that encodes the constraints of a view. If a bit in the long integer is set, the current view exhibits the corresponding constraint. The bits are addressed using these constants:

```
enum {
    kQTVRUnconstrained                  = 0
    kQTVRCantPanLeft                    = 1L << 0,
    kQTVRCantPanRight                   = 1L << 1,
    kQTVRCantPanUp                      = 1L << 2,
    kQTVRCantPanDown                    = 1L << 3,
    kQTVRCantZoomIn                     = 1L << 4,
    kQTVRCantZoomOut                    = 1L << 5,
    kQTVRCantTranslateLeft              = 1L << 6,
    kQTVRCantTranslateRight             = 1L << 7,
    kQTVRCantTranslateUp                = 1L << 8,
    kQTVRCantTranslateDown              = 1L << 9
};
```

Constant descriptions

`kQTVRUnconstrained` The view has no constraints. This is not a bit selector but a value that indicates that none of the following bits is set.

`kQTVRCantPanLeft` If this bit is set, the view cannot pan left.

`kQTVRCantPanRight` If this bit is set, the view cannot pan right.

`kQTVRCantPanUp` If this bit is set, the view cannot pan up.

`kQTVRCantPanDown` If this bit is set, the view cannot pan down.

`kQTVRCantZoomIn` If this bit is set, the view cannot zoom in.

`kQTVRCantZoomOut` If this bit is set, the view cannot zoom out.

`kQTVRCantTranslateLeft`
If this bit is set, the view cannot translate to the left. This constraint is valid only for object nodes.

`kQTVRCantTranslateRight`
If this bit is set, the view cannot translate to the right. This constraint is valid only for object nodes.

`kQTVRCantTranslateUp`
If this bit is set, the view cannot translate up. This constraint is valid only for object nodes.

`kQTVRCantTranslateDown`
If this bit is set, the view cannot translate down. This constraint is valid only for object nodes.

Mouse Control Modes

The value returned by the `QTVRGetCurrentMouseMode` function is an unsigned long integer that encodes the current mouse control modes. If a bit in the integer is set, the corresponding mode is one of the current mouse modes. The mode bits are addressed using these constants:

```
enum {
    kQTVRPanning                                = 1L << 0,
    kQTVRTranslating                            = 1L << 1,
    kQTVRZooming                                = 1L << 2,
    kQTVRScrolling                              = 1L << 3,
    kQTVRSelecting                              = 1L << 4
};
```

Notice that several modes can be returned. That means a return value could have both zooming and translating set, for example.

Constant descriptions

`kQTVRPanning` If this bit is set, the mouse controls panning of standard objects, using objects-only controllers.

kQTVRTranslating	If this bit is set, the mouse controls translation for all objects.
kQTVRZooming	If this bit is set, the mouse controls zooming for all objects.
kQTVRScrolling	If this bit is set, the mouse controls arrow scrolling for standard objects and scrolling for joystick objects.
kQTVRSelecting	If this bit is set, the mouse controls selecting of objects as an object absolute controller.

Hot Spot Selectors

The enableFlag parameter passed to the function QTVREnableHotSpot (page 108) indicates the method you want to use to select one or more hot spots for enabling or disabling. These constants define the available hot spot selectors:

```
enum {
    kQTVRHotSpotID                          = 0,
    kQTVRHotSpotType                        = 1,
    kQTVRAllHotSpots                        = 2
};
```

Constant descriptions

kQTVRHotSpotID	Select a single hot spot by its hot spot ID (specified by the hotSpotValue parameter).
kQTVRHotSpotType	Select all hot spots of a hot spot type (specified by the hotSpotValue parameter).
kQTVRAllHotSpots	Select all hot spots in the current node.

Note

These hot spot selectors are not masks. So, for example, if you want to enable only hot spots of a certain type, you must first disable all hot spots (using the kQTVRAllHotSpots selector) and then enable the desired type of hot spots (using the kQTVRHotSpotType selector). ◆

Animation Settings

The setting parameter passed to the QTVRGetAnimationSetting and QTVRSetAnimationSetting functions is a long integer that specifies a particular animation setting of an object node. Animation settings specify characteristics

of the movie while it is playing. You can use these constants to specify animation settings:

```
typedef enum QTVRObjectAnimationSetting {
    kQTVRPalindromeViewFrames                   = 1,
    kQTVRStartFirstViewFrames                   = 2,
    kQTVRDontLoopViewFrames                     = 3,
    kQTVRPlayEveryViewFrame                     = 4,
    kQTVRSyncViewToFrameRate                    = 16,
    kQTVRPalindromeViews                        = 17
    kQTVRPlayStreamingViews                     = 18
} QTVRObjectAnimationSetting;
```

Constant descriptions

`kQTVRPalindromeViewFrames`

Play a back-and-forth animation of the frames of the current view. The frames of the current view play with a positive or negative frame rate; the frame rate sign is switched each time the view end time (equal to the view duration) or the view start time (always 0) is reached.

`kQTVRStartFirstViewFrames`

Play the frame animation starting with the first frame in the view (that is, at the view start time). This setting is useful if a sound track is associated by time with object views. Even if the object view contains no animation, setting this flag allows any sound authored to play with the view to play from the beginning. When this flag is clear, each new object view begins playing using the current view time of the previous view.

`kQTVRDontLoopViewFrames`

Don’t loop the frame animation. Animation frames and sound stop playing when the view duration is reached.

`kQTVRPlayEveryViewFrame`

Play every view frame. Animation plays all frames regardless of play rate. The play rate is used to adjust the duration in which a frame appears but no frames are skipped so the rate is not exact. When this property is set, sound tracks are not played.

`kQTVRSyncViewToFrameRate`

Synchronize the view animation to the frame animation.

When view animation is enabled, the object views play at the same rate and animation settings as the frame animation rate and settings. This is useful if animation must be synchronized precisely across multiple views or a sound track is to be played during view animation instead of during frame animation.

`kQTVRPalindromeViews`

Play a back-and-forth animation of the views of the current node. When view animation is enabled, the object views play with a positive or negative view rate; the view rate sign is switched each time an object's pan equals the object's minimum pan limit or the object's maximum pan limit (the first column in a row or the last column in a row, respectively).

`kQTVRPlayStreamingViews`

When an object movie is streaming in from a network, this flag indicates whether the frames corresponding to the row in which the default view appears are to be played as they are loaded.

Control Settings

The `setting` parameter passed to the `QTVRGetControlSetting` and `QTVRSetControlSetting` functions is a long integer that specifies a particular control setting of an object node. Control settings specify whether the object can wrap during panning and tilting, as well as other features of the node. The control settings are specified using these constants:

```
typedef enum QTVRControlSetting {
    kQTVRWrapPan                                = 1,
    kQTVRWrapTilt                               = 2,
    kQTVRCanZoom                                = 3,
    kQTVRReverseHControl                        = 4,
    kQTVRReverseVControl                        = 5,
    kQTVRSwapHVControl                          = 6,
    kQTVRTranslation                            = 7
} QTVRControlSetting;
```

Constant descriptions

`kQTVRWrapPan` Set wrapping during panning. When this control setting is enabled, the user can wrap around from the current pan constraint maximum value to the pan constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, calling the `QTVRSetPanAngle` function with a pan angle that is either less than or greater than the current pan constraints results in a pan angle within the current pan constraints. Similarly, `QTVRWrapAndConstrain` returns a pan angle within the current pan constraints, and `QTVRNudge` wraps views after it reaches the maximum or minimum constraint value.

When this control setting is disabled, the user cannot wrap around from the current pan constraint maximum value to the pan constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, the `QTVRSetPanAngle` function returns the result code `constraintReachedErr` when passed a pan angle that is either less than or greater than the current pan constraints. Similarly, `QTVRWrapAndConstrain` returns a pan angle that is one of the current pan constraints, and `QTVRNudge` returns the result code `constraintReachedErr` when it reaches the maximum or minimum constraint value.

Note that a view animation stops when a constraint is reached unless palindrome view animation or wrapping during panning is enabled.

`kQTVRWrapTilt` Set wrapping during tilting. When this control setting is enabled, the user can wrap around from the current tilt constraint maximum value to the tilt constraint minimum value (or vice versa) using the mouse or arrow keys. In addition, calling the `QTVRSetTiltAngle` function with a tilt angle that is either less than or greater than the current tilt constraints results in a tilt angle within the current tilt constraints. Similarly, `QTVRWrapAndConstrain` returns a tilt angle within the current tilt constraints, and `QTVRNudge` wraps views after it reaches the maximum or minimum constraint value.

When this control setting is disabled, the user cannot wrap around from the current tilt constraint maximum value to the tilt constraint minimum value (or vice versa) using the

mouse or arrow keys. In addition, the `QTVRSetTiltAngle` function returns the result code `constraintReachedErr` when passed a tilt angle that is either less than or greater than the current tilt constraints. Similarly, `QTVRWrapAndConstrain` returns a tilt angle that is one of the current tilt constraints, and `QTVRNudge` returns the result code `constraintReachedErr` when it reaches the maximum or minimum constraint value.

`kQTVRCanZoom`

Set zooming to be active. When this control setting is enabled, the user can change the current field of view using the zoom-in and zoom-out keys on the keyboard (or using the VR controller buttons). In addition, you can use the `QTVRSetFieldOfView` function to alter the current field of view. Similarly, `QTVRWrapAndConstrain` returns a field of view within the current field of view constraints.

When this control is disabled, the user cannot change the current field of view using the zoom-in and zoom-out keys on the keyboard (or using the VR controller buttons). In addition, the `QTVRSetFieldOfView` function returns the result code `constraintReachedErr` and doesn't change the field of view. Similarly, `QTVRWrapAndConstrain` returns the current field of view.

`kQTVRReverseHControl`

Reverse the direction of the horizontal control. When this control setting is enabled, the user can change an object's horizontal view using the mouse or the keyboard arrows with reversed values for mouse horizontal motion and keyboard left and right arrows. (In other words, the left arrow key causes panning to the right, and moving the mouse right causes panning to the left.) Similarly, the `QTVRNudge` function nudges left when you pass the value 0 and right when you pass the value 180. This control setting is useful when an object's frames have been authored in reverse order.

`kQTVRReverseVControl`

Reverse the direction of the vertical control. When this control setting is enabled, the user can change an object's vertical view using the mouse or the keyboard arrows with reversed values for mouse vertical motion and keyboard up and down arrows. (In other words, the up arrow key

causes tilting down, and moving the mouse down causes tilting up.) Similarly, the `QTVRNudge` function nudges up when you pass the value 270 and down when you pass the value 90. This control setting is useful when an object's frames have been authored in reverse order.

`kQTVRSwapHVControl` Exchange the horizontal and vertical controls. When this setting is enabled, the user can pan left using the up arrow key and tilt up using the left arrow key. Similarly, the `QTVRNudge` function nudges up when you pass the value 180 and down when you pass the value 0. This control setting is useful if an object is a single-row or multirow movie containing vertically changing images. This is especially useful when enabling view animation for an object with animating vertical changes in view (because objects will animate only along rows, not along columns).

`kQTVRTranslation` Set translation to be active. When this setting is enabled, the user can translate using the mouse when either the translation key is held down or the controller translation mode button is toggled on. In addition, you can use the `QTVRSetViewCenter` function to set a horizontal and vertical position in the current display coordinate system.

When this setting is disabled, the `QTVRWrapAndConstrain` function always returns the current view center, and the `QTVRSetViewCenter` function returns the result code `constraintReachedErr` and doesn't change the current view center.

View State Types

The `viewStateType` parameter to the `QTVRGetViewState` and `QTVRSetViewState` functions specifies a view state type. The QuickTime VR Manager defines the following constants for view state types:

```
typedef enum QTVRViewStateType {
    kQTVRDefault                                = 0,
    kQTVRCurrent                                = 2,
    kQTVRMouseDown                              = 3
} QTVRViewStateType;
```

Constant descriptions

`kQTVRDefault`	The default view state of the current view. The default view state is a set of object images defined by view duration, row, and column. The default view state image for a given view is displayed when the mouse button is not down.
`kQTVRCurrent`	The current view state of the current view. The current view state can be any of the view states authored in an object movie. Setting the current view state does not change the view state designated as the `kQTVRDefault` or `kQTVRMouseDown` view state. The effect of changing the current view state lasts until a transition to the mouse-down or default view state occurs.
`kQTVRMouseDown`	The mouse-down state of the current view. The mouse-down view state is a set of object images defined by view duration, row, and column. The mouse-down view state image for a given view is displayed when the user holds the mouse button down while the cursor is over an object movie.

Back Buffer Imaging Procedure Flags

The QuickTime VR Manager defines three sets of flags for use in connection with a back buffer imaging procedure: (1) a set of flags for the `flags` field in an area of interest structure passed to the `QTVRSetBackBufferImagingProc` function, (2) a set of flags for the `flagsIn` parameter to a back buffer imaging procedure, and (3) a set of flags for the `flagsOut` parameter to a back buffer imaging procedure.

The `flags` field in an area of interest structure (page 87) passed to the function `QTVRSetBackBufferImagingProc` (page 177) is a long integer whose bits indicate when to call the back buffer imaging procedure for the specified area of interest. If a bit in the long integer is set, the back buffer imaging procedure is called at the corresponding time. The bits are addressed using these constants:

```
enum {
    kQTVRBackBufferEveryUpdate                  = 1L << 0,
    kQTVRBackBufferEveryIdle                    = 1L << 1,
    kQTVRBackBufferAlwaysRefresh                = 1L << 2
};
```

Constant descriptions

`kQTVRBackBufferEveryUpdate`
: If this bit is set, the back buffer imaging procedure is to be called whenever QuickTime VR is about to update the window containing the specified QuickTime VR movie instance. That is, the procedure is called just before QuickTime VR unwarps the back buffer image into the prescreen buffer and redraws the screen image.

`kQTVRBackBufferEveryIdle`
: If this bit is set, the back buffer imaging procedure is to be called when either `MCIsPlayerEvent(idle)` or `MCIdle` is called). Its purpose is to cause the software to draw as often as possible.

`kQTVRBackBufferAlwaysRefresh`
: If this bit is set, the back buffer is always refreshed to the proper movie data just before your back buffer imaging procedure is called. If your back buffer imaging procedure completely overwrites the rectangle passed to it, you should not set this bit.

The `flagsIn` parameter passed to a back buffer imaging procedure specifies the event or operation that caused your procedure to be called, as well as other information about the state of the back buffer when your procedure is called. The bits in the `flagsIn` parameter are addressed using these constants:

```
enum {
    kQTVRBackBufferRectVisible                  = 1L << 0,
    kQTVRBackBufferWasRefreshed                 = 1L << 1
};
```

Constant descriptions

`kQTVRBackBufferRectVisible`
: If this bit is set, the specified rectangle is currently visible. Your back buffer imaging procedure is always called at least once with this bit set, when the rectangle first becomes visible. When the rectangle becomes no longer visible, your back buffer imaging procedure is called with this bit clear.

`kQTVRBackBufferWasRefreshed`
: If this bit is set, QuickTime VR refreshed the back buffer prior to calling your back buffer imaging procedure.

Before returning from your back buffer imaging procedure, you should set the `flagsOut` parameter to indicate what actions you performed in your procedure. You can set bits in that parameter using this constant:

```
enum {
    kQTVRBackBufferFlagDidDraw                      = 1L << 0
};
```

Constant description

`kQTVRBackBufferFlagDidDraw`
Set this bit if your back buffer imaging procedure performed any drawing in the back buffer.

Nudge Mode

The `kQTVRInteractionNudgeMode` property is used with the functions `QTVRGetInteractionProperty` (page 163) and `QTVRSetInteractionProperty` (page 164) and specifies the type of action to be taken by a nudge. You can use these constants to specify the nudge mode:

```
typedef enum QTVRNudgeMode {
    kQTVRNudgeRotate                                = 0UL,
    kQTVRNudgeTranslate                             = 1UL,
    kQTVRUNudgeSameAsMouse                          = 2UL
} QTVRNudgeMode;
```

Constant descriptions

`kQTVRNudgeRotate` Set the nudge mode to rotate the object.

`kQTVRNudgeTranslate` Set the nudge mode to translate the image.

`kQTVRUNudgeSameAsMouse` Set the nudge mode to the same as the current mouse mode.

Nudge Directions

The `direction` parameter to the function `QTVRNudge` (page 98) specifies the direction in which to nudge the current view. You can use these constants to specify a nudge direction:

```
typedef enum QTVRNudgeControl {
    kQTVRRight                                  = 0,
    kQTVRUpRight                                = 45,
    kQTVRUp                                     = 90,
    kQTVRUpLeft                                 = 135,
    kQTVRLeft                                   = 180,
    kQTVRDownLeft                               = 225,
    kQTVRDown                                   = 270,
    kQTVRDownRight                              = 315
} QTVRNudgeControl;
```

IMPORTANT

The actual direction of the nudge is affected by the current control settings. For example, when the `kQTVRReverseVControl` control setting is enabled, the `QTVRNudge` function nudges up when you pass the value `kQTVRDown` and down when you pass the value `kQTVRUp`. ▲

Constant descriptions

`kQTVRRight`	Nudge the current view to the right.
`kQTVRUpRight`	Nudge the current view up and to the right.
`kQTVRUp`	Nudge the current view up.
`kQTVRUpLeft`	Nudge the current view up and to the left.
`kQTVRLeft`	Nudge the current view to the left.
`kQTVRDownLeft`	Nudge the current view down and to the left.
`kQTVRDown`	Nudge the current view down.
`kQTVRDownRight`	Nudge the current view down and to the right.

Cursor Types

The `type` field of a cursor record (page 87) specifies the kind of cursor you want to replace (or restore) using the function `QTVRReplaceCursor` (page 165). You can use these constants to specify a cursor type:

```
enum {
    kQTVRUseDefaultCursor                   = 0,
    kQTVRStdCursorType                      = 1,
    kQTVRColorCursorType                    = 2
};
```

Constant descriptions

`kQTVRUseDefaultCursor`
Restore the default cursor. In this case, the `handle` field of the cursor record should be `nil`.

`kQTVRStdCursorType` The cursor is a standard black-and-white cursor.

`kQTVRColorCursorType`
The cursor is a color cursor.

Pixel Formats

QuickTime VR 2.1 supports imaging to and from buffers in any of several pixel formats, allowing for imaging directly to the screen buffer on most video cards used in computers that support Microsoft Windows. The `cachePixelFormat` parameter passed to the functions `QTVRGetBackBufferMemInfo` (page 171), `QTVRGetBackBufferSettings` (page 173), and `QTVRSetBackBufferPrefs` (page 174) specifies the pixel format.

On the Macintosh, QuickTime VR 2.1 can directly image into the standard 16- and 32-bit big-endian formats (16BE555 and 32ARGB). On Windows systems, QuickTime VR 2.1 supports the Macintosh formats along with 16LE555, 16LE565, 32BGRA, 32ABGR, 32RGBA, 24RGB and 24BGR.

The pixel formats are defined as follows:

```
#define k16LE555PixelFormat     FOUR_CHAR_CODE('L555')   // 16 bit LE rgb 555 (PC)
#define k16BE565PixelFormat     FOUR_CHAR_CODE('B565')   // 16 bit BE rgb 565
#define k16LE565PixelFormat     FOUR_CHAR_CODE('L565')   // 16 bit LE rgb 565
#define k24BGRPixelFormat       FOUR_CHAR_CODE('24BG')   // 24bit bgr
#define k32ARGBPixelFormat      0x00000020               // 32 bit argb
```

```
#define k32BGRAPixelFormat      FOUR_CHAR_CODE('BGRA')   // 32 bit bgra (Matrox)
#define k32ABGRPixelFormat      FOUR_CHAR_CODE('ABGR')   // 32 bit abgr
#define k32RGBAPixelFormat      FOUR_CHAR_CODE('RGBA')   // 32 bit rgba
```

Note
Developers should allow for the possibility that future versions of QuickTime VR may support additional formats. For example, an application can try the preferred format first and then use one of the listed values if the first attempt returns an error. ◆

You can use the pixel format constants defined above in the `cachePixelFormat` parameter passed to the `QTVRGetBackBufferMemInfo`, `QTVRGetBackBufferSettings`, and `QTVRSetBackBufferPrefs` functions, or you may use the following constants to specify only the depth:

```
enum{
    kQTVRUseMovieDepth                          = 0,
};
```

Constant descriptions

`kQTVRUseMovieDepth` A pixel has the same depth as the movie window.

When performing direct-to-screen drawing of panoramas, the cache format must match the screen format. Otherwise, the panoramic image will be generated offscreen and then copied to the screen. If the application has not set a cache format preference, QuickTime VR 2.1 tries to match the backbuffer format to the screen format.

If the application has installed a prescreen buffer imaging complete procedure, the GWorld passed to it will be in the same format as the panoramic backbuffer.

Resolutions

The `resolution` parameter to the `QTVRGetBackBufferMemInfo`, `QTVRGetBackBufferSettings`, and `QTVRSetBackBufferPrefs` functions specifies the resolution of an image. You can use these constants to specify a resolution:

```
enum {
    kQTVRDefaultRes                             = 0,
    kQTVRFullRes                                = 1L << 0,
```

```
    kQTVRHalfRes                                = 1L << 1,
    kQTVRQuarterRes                             = 1L << 2
};
```

Constant descriptions

`kQTVRDefaultRes`	The default resolution of the image.
`kQTVRFullRes`	The full resolution of the image.
`kQTVRHalfRes`	One-half the full resolution of the image.
`kQTVRQuarterRes`	One-quarter the full resolution of the image.

Geometry Selectors

The `geometry` parameter used in the procedures `QTVRSetBackBufferPrefs`, `QTVRGetBackBufferSettings`, and `QTVRGetBackBufferMemInfo` specifies the type and orientation of the panorama data. Only the vertical cylinder geometry is used in the current version of QuickTime VR; calls to `QTVRGetBackBufferSettings` and `QTVRGetBackBufferMemInfo` always return the value `kQTVRVerticalCylinder`. The parameter is present to provide a means of expansion to new panoramic geometries in the future.

```
enum {
    kQTVRUseMovieGeometry                       = 0,
    kQTVRVerticalCylinder                       = 'vcyl'
};
```

For a description of the orientation of the panorama data, see page 275 in Chapter 5, “Creating QuickTime VR Movies.”

Cache Sizes

The `cacheSize` parameter to the `QTVRGetBackBufferSettings` and `QTVRSetBackBufferPrefs` functions specifies the size of the panorama back buffer. You can use these constants to specify a cache size:

```
enum {
    kQTVRMinimumCache                           = -1,
    kQTVRSuggestedCache                         = 0,
    kQTVRFullCache                              = 1
};
```

Constant descriptions

`kQTVRMinimumCache`	The minimum cache size required to display the specified movie.
`kQTVRSuggestedCache`	The suggested cache size, a cache large enough to allow full zooming out of the panorama.
`kQTVRFullCache`	The full cache size (that is, a cache that is large enough to fit the entire panorama in memory). This is the default cache size.

Data Structures

This section describes the data structures provided by the QuickTime VR Manager.

Intercept Structure

When QuickTime VR calls an intercept procedure, it passes the procedure information about the intercepted function and the parameters for that function in an intercept record. An intercept record is defined by the `QTVRInterceptRecord` data type:

```
typedef UInt32              QTVRProcSelector;
struct QTVRInterceptRecord {
    SInt32                      reserved1;
    SInt32                      selector;
    SInt32                      reserved2;
    SInt32                      reserved3;

    SInt32                      paramCount;
    void *                      parameter[6];
};
typedef struct QTVRInterceptRecord QTVRInterceptRecord;
typedef QTVRInterceptRecord *QTVRInterceptPtr;
```

Field descriptions

`reserved1`	Reserved for use by Apple Computer, Inc.

selector	A selector that indicates which QuickTime VR Manager function has been intercepted. See “Intercept Selectors” (page 60) for a description of the available intercept selectors.
reserved2	Reserved for use by Apple Computer, Inc.
reserved3	Reserved for use by Apple Computer, Inc.
paramCount	The number of parameters contained in the array pointed to by the parameter field.
parameter	An array of the parameters for the QuickTime VR function specified by the selector field. The order of parameters in this array matches the order of the parameters passed to the specified function, except that the QuickTime VR instance parameter is *not* included in this array.

IMPORTANT

The QuickTime VR Manager internally stores all angle measurements in radians, and any angular parameters returned in the parameter field of an intercept structure are expressed in radians. In addition, a parameter of type float (for example, the panAngle parameter to the QTVRSetPanAngle function) is passed in the parameter field as a value of type *float. ▲

Floating-Point Point Structure

Several QuickTime VR Manager functions use a **floating-point point structure** to specify a point in a panorama or an object. The floating-point point structure is defined by the QTVRFloatPoint data type:

```
struct QTVRFloatPoint {
    float                       x;
    float                       y;
};
typedef struct QTVRFloatPoint QTVRFloatPoint;
```

Field descriptions

x	The horizontal coordinate of the point.
y	The vertical coordinate of the point.

Cursor Record

The `cursRecord` parameter to the function `QTVRReplaceCursor` (page 165) specifies a **cursor record,** which indicates the cursor to replace and its replacement cursor. A cursor record is defined by the `QTVRCursorRecord` data type:

```
struct QTVRCursorRecord {
    UInt16                      theType;        /* field was previously
                                                    named "type"*/
    SInt16                      rsrcID;
    Handle                      handle;
};
typedef struct QTVRCursorRecord QTVRCursorRecord;
```

Field descriptions

`theType` The type of cursor to replace. The available cursor types are defined by constants; see "Cursor Types" (page 82).

`rsrcID` The resource ID of the cursor to replace. See the appendix, "QuickTime VR Cursors," for a list of the currently used cursor resource IDs.

`handle` A handle to the cursor data that is to replace the specified cursor. If `theType` is `kQTVRUseDefaultCursor`, then this field should contain the value `nil`.

Area of Interest Structure

The `areasOfInterest` parameter to the function `QTVRSetBackBufferImagingProc` (page 177) specifies an array of **area of interest structures,** each one of which indicates a rectangular area in the back buffer. An area of interest structure is defined by the `QTVRAreaOfInterest` data type:

```
struct QTVRAreaOfInterest {
    float                       panAngle;
    float                       tiltAngle;
    float                       width;
    float                       height;
    UInt32                      flags;
};
typedef struct QTVRAreaOfInterest QTVRAreaOfInterest;
```

Field descriptions

`panAngle`	The pan angle of the upper-left coordinate (in panorama space) of the area of interest.
`tiltAngle`	The tilt angle of the upper-left coordinate (in panorama space) of the area of interest.
`width`	The width of the area of interest.
`height`	The height of the area of interest.
`flags`	A set of bit flags that indicate when to call the back buffer imaging procedure for this area of interest. See “Back Buffer Imaging Procedure Flags” (page 78) for a description of the available flags.

QuickTime VR Manager Routines

This section describes the routines provided by the QuickTime VR Manager.

Initializing and Terminating QuickTime VR

The QuickTime VR Manager provides routines for initializing and terminating its operation.

Note
The `InitializeQTVR` and `TerminateQTVR` routines are required for QuickTime VR to run in a Windows environment. They do nothing in the Mac OS environment, but should be included for cross-platform compatibility.

When your application calls `InitializeQTVR` in the Windows environment, the code attempts to find `QuickTimeVR.qtx` through the normal search paths. If it does not find `QuickTimeVR.qtx`, it returns an error code of 6660 and the API will be unusable.

InitializeQTVR

You must use the `InitializeQTVR` function before calling other functions of QuickTime VR. The `InitializeQTVR` function tries to load the `QuickTimeVR.qtx`

file, first from the application directory and then from the system directory. If the `InitializeQTVR` function is unable to load the `QuickTimeVR.qtx` file, it returns error code qtvrLibraryLoadErr (–30554).

If you call any other function of QuickTime VR before calling the `InitializeQTVR` function or after the `InitializeQTVR` function has failed to load the `QuickTimeVR.qtx` file, QuickTime VR returns error code qtvrUninitialized (–30555).

TerminateQTVR

You must use the `TerminateQTVR` function when you have finished using the functions of QuickTime VR.

Multiple calls to InitializeQTVR and TerminateQTVR can be either nested or sequential, but there must be at least one call to TerminateQTVR corresponding to each call to InitializeQTVR.

Initializing and Managing QuickTime VR Movie Instances

The QuickTime VR Manager provides routines for obtaining a QTVR track and a new movie instance.

QTVRGetQTVRTrack

You can use the `QTVRGetQTVRTrack` function to get a QTVR track contained in a QuickTime movie to use in the `QTVRGetQTVRInstance` call.

```
Track QTVRGetQTVRTrack (Movie theMovie, SInt32 index);
```

`theMovie`	A QuickTime movie.
`index`	The index of the desired QTVR track.
function result	A track identifier for the QTVR track having the specified index in the specified QuickTime movie.

DESCRIPTION

The `QTVRGetQTVRTrack` function returns, as its function result, a track identifier for the QTVR track that has the index specified by the `index` parameter in the QuickTime movie specified by the `theMovie` parameter. If there is no such track, `QTVRGetQTVRTrack` returns the value `nil`.

SPECIAL CONSIDERATIONS

QuickTime VR 2.1 supports files with at most one QTVR track, hence the value for the index parameter should always be one. Future versions may support multiple QTVR tracks per file.

Panorama and object movies made with QuickTime VR version 1.0 have no QTVR track. The `QTVRGetQTVRTrack` function returns the track ID of the panorama track for version 1.0 panorama movies and the track ID of the image video track for version 1.0 object movies. For non-QTVR movies, the `QTVRGetQTVRTrack` function returns nil.

SEE ALSO

Use `QTVRGetQTVRInstance` (next) to get a QuickTime VR movie instance from the track identifier returned by `QTVRGetQTVRTrack`.

QTVRGetQTVRInstance

You can use the `QTVRGetQTVRInstance` function to get an instance of a QuickTime VR movie.

```
OSErr QTVRGetQTVRInstance (
                    QTVRInstance *qtvr,
                    Track qtvrTrack,
                    MovieController mc);
```

`qtvr` On exit, an instance of the specified QuickTime VR movie.

`qtvrTrack` A QTVR track contained in a QuickTime movie. You can obtain a reference to this track by calling `QTVRGetQTVRTrack` (page 89).

`mc`	An identifier for the movie controller to be associated with the new QuickTime VR movie instance. You can get a movie controller identifier by calling the `NewMovieController` function.
function result	A result code.

DESCRIPTION

The `QTVRGetQTVRInstance` function returns, in the `qtvr` parameter, an instance of the QuickTime VR movie specified by the `qtvrTrack` parameter. If `qtvrTrack` does not specify a QTVR track, `QTVRGetQTVRInstance` returns `nil` in the `qtvr` parameter and an error code as its function result. You need a QuickTime VR movie instance to call most other QuickTime VR functions.

SPECIAL CONSIDERATIONS

It's not necessary to dispose of a QuickTime VR movie instance.

Manipulating Viewing Angles and Zooming

The QuickTime VR Manager provides functions that you can use to manipulate the viewing angles and zooming characteristics of a QuickTime VR movie. Note that the parameters to these functions that specify angles are always interpreted in the current angular unit (degrees or radians) set by a previous call to the `QTVRSetAngularUnits` function. The default angular unit is degrees.

QTVRGetPanAngle

You can use the `QTVRGetPanAngle` function to get the pan angle of a QuickTime VR movie.

```
float QTVRGetPanAngle (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A floating-point value that represents the current pan angle of the specified movie.

DESCRIPTION

The `QTVRGetPanAngle` function returns, as its function result, a floating-point value that represents the current pan angle of the QuickTime VR movie specified by the `qtvr` parameter.

SEE ALSO

Use `QTVRSetPanAngle` (next) to set the pan angle of a movie. Listing 2-3 (page 46) illustrates the use of `QTVRGetPanAngle`.

QTVRSetPanAngle

You can use the `QTVRSetPanAngle` function to set the pan angle of a QuickTime VR movie.

```
OSErr QTVRSetPanAngle (QTVRInstance qtvr, float panAngle);
```

`qtvr`	An instance of a QuickTime VR movie.
`panAngle`	The desired pan angle of the specified movie.
function result	A result code.

DESCRIPTION

The `QTVRSetPanAngle` function sets the pan angle of the QuickTime VR movie specified by the `qtvr` parameter to the value specified by the `panAngle` parameter. That value is constrained by the maximum and minimum pan angles of the movie. If the angle falls outside of those constraints and the control setting `kQTVRWrapPan` is disabled, the angle is set to the minimum or maximum, whichever is closer. If wrapping is enabled, the pan angle is clipped to fall within the constraints. Pan angle values are also clipped if the requested pan angle, when combined with the current tilt angle and field of view, would cause an image to lie outside the current constraints.

`QTVRSetPanAngle` returns the result code `constraintReachedErr` if wrapping is off and the angle is set to the minimum or maximum constraint value.

SPECIAL CONSIDERATIONS

The pan and tilt angles are subject to the current pan and tilt range constraints, as imposed by the viewing limits and the current field of view. Accordingly, if you want to change the field of view, you should do so before adjusting the pan or tilt angles. Otherwise, the pan and tilt angles are clipped against the current field of view, which may result in an incorrect view when you alter the field of view.

SEE ALSO

Use `QTVRGetPanAngle` (page 91) to get the pan angle of a movie. Use `QTVRGetViewingLimits` (page 166) to get the current viewing limits of a movie. Listing 2-3 (page 46) illustrates the use of `QTVRSetPanAngle`. Use `QTVRSetControlSetting` (page 140) to control the setting of `kQTVRWrapPan`.

QTVRGetTiltAngle

You can use the `QTVRGetTiltAngle` function to get the tilt angle of a QuickTime VR movie.

```
float QTVRGetTiltAngle (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A floating-point value that represents the current tilt angle of the specified movie.

DESCRIPTION

The `QTVRGetTiltAngle` function returns, as its function result, a floating-point value that represents the current tilt angle of the QuickTime VR movie specified by the `qtvr` parameter.

SEE ALSO

Use `QTVRSetTiltAngle` (next) to set the tilt angle of a movie. Listing 2-3 (page 46) illustrates the use of `QTVRGetTiltAngle`.

QTVRSetTiltAngle

You can use the `QTVRSetTiltAngle` function to set the tilt angle of a QuickTime VR movie.

```
OSErr QTVRSetTiltAngle (QTVRInstance qtvr, float tiltAngle);
```

`qtvr`	An instance of a QuickTime VR movie.
`tiltAngle`	The desired tilt angle of the specified movie.
function result	A result code.

DESCRIPTION

The `QTVRSetTiltAngle` function sets the tilt angle of the QuickTime VR movie specified by the `qtvr` parameter to the value specified by the `tiltAngle` parameter. That value is constrained by the maximum and minimum tilt angles of the movie. If the angle falls outside of those constraints and the control setting `kQTVRWrapTilt` is disabled, the angle is set to the minimum or maximum, whichever is closer. If wrapping is enabled, the tilt angle is clipped to fall within the constraints. Tilt angle values are also clipped if the requested tilt angle, when combined with the current pan angle and field of view, would cause an image to lie outside the current constraints.

`QTVRSetTiltAngle` returns the result code `constraintReachedErr` if wrapping is off and the angle is set to the minimum or maximum constraint value.

SPECIAL CONSIDERATIONS

The pan and tilt angles are subject to the current pan and tilt range constraints, as imposed by the viewing limits and the current field of view. Accordingly, if you want to change the field of view, you should do so before adjusting the pan or tilt angles. Otherwise, the pan and tilt angles are clipped against the current field of view, which may result in an incorrect view when you alter the field of view.

SEE ALSO

Use `QTVRGetTiltAngle` (page 93) to get the tilt angle of a movie. Use `QTVRGetViewingLimits` (page 166) to get the current viewing limits of a movie.

Listing 2-3 (page 46) illustrates the use of `QTVRSetTiltAngle`. Use `QTVRSetControlSetting` (page 140) to control the setting of `kQTVRWrapTilt`.

QTVRGetFieldOfView

You can use the `QTVRGetFieldOfView` function to get the vertical field of view of a QuickTime VR movie.

```
float QTVRGetFieldOfView (QTVRInstance qtvr);
```

`qtvr` An instance of a QuickTime VR movie.

function result The current vertical field of view of the specified movie.

DESCRIPTION

The `QTVRGetFieldOfView` function returns, as its function result, the current vertical field of view of the QuickTime VR movie specified by the `qtvr` parameter. The vertical field of view is a floating-point value that specifies the angle created by the two lines that connect the viewpoint to the top and bottom of the image.

SEE ALSO

Use `QTVRSetFieldOfView` (next) to set the vertical field of view of a QuickTime VR movie. Listing 2-4 (page 48) illustrates the use of `QTVRGetFieldOfView`.

QTVRSetFieldOfView

You can use the `QTVRSetFieldOfView` function to set the vertical field of view of a QuickTime VR movie.

```
OSErr QTVRSetFieldOfView (QTVRInstance qtvr, float fieldOfView);
```

`qtvr` An instance of a QuickTime VR movie.

`fieldOfView` The desired vertical field of view for the specified movie.

function result A result code.

DESCRIPTION

The `QTVRSetFieldOfView` function sets the vertical field of view of the QuickTime VR movie specified by the `qtvr` parameter to the value specified by the `fieldOfView` parameter. That value is constrained by the maximum field of view of the movie. Values that lie outside that limit are clipped to the maximum. Pan and tilt angle values are also clipped if, when combined with the current field of view, they would cause an image to lie outside the current constraints.

If the control setting `kQTVRCanZoom` is disabled, the field of view is unchanged and `QTVRSetFieldOfView` returns the result code `constraintReachedErr`.

SPECIAL CONSIDERATIONS

The pan and tilt angles are subject to the current pan and tilt range constraints, as imposed by the viewing limits and the current field of view. Accordingly, if you want to change the field of view, you should do so before adjusting the pan or tilt angles. Otherwise, the pan and tilt angles are clipped against the current field of view, which may result in an incorrect view when you alter the field of view.

SEE ALSO

Use `QTVRGetFieldOfView` (page 95) to get the vertical field of view of a QuickTime VR movie. Listing 2-4 (page 48) illustrates the use of `QTVRSetFieldOfView`. Use `QTVRSetControlSetting` (page 140) to control the setting of `kQTVRCanZoom`.

QTVRGetViewCenter

You can use the `QTVRGetViewCenter` function to get the view center of a QuickTime VR movie.

```
OSErr QTVRGetViewCenter (QTVRInstance qtvr, QTVRFloatPoint *viewCenter);
```

`qtvr`	An instance of a QuickTime VR movie.
`viewCenter`	On entry, a pointer to a `QTVRFloatPoint` structure. On exit, that structure contains the current view center of the specified movie.
function result	A result code.

DESCRIPTION

The `QTVRGetViewCenter` function returns, in the `QTVRFloatPoint` structure pointed to by the `viewCenter` parameter, the *x* and *y* coordinates of the current view center of the QuickTime VR movie specified by the `qtvr` parameter.

SPECIAL CONSIDERATIONS

`QTVRGetViewCenter` is valid only for object nodes.

SEE ALSO

Use `QTVRSetViewCenter` (next) to set the view center of a movie.

QTVRSetViewCenter

You can use the `QTVRSetViewCenter` function to set the view center of a QuickTime VR movie.

```
OSErr QTVRSetViewCenter (
                    QTVRInstance qtvr,
                    const QTVRFloatPoint *viewCenter);
```

`qtvr`	An instance of a QuickTime VR movie.
`viewCenter`	A pointer to a `QTVRFloatPoint` structure that contains the desired view center of the specified movie.
function result	A result code.

DESCRIPTION

The `QTVRSetViewCenter` function sets the view center of the QuickTime VR movie specified by the `qtvr` parameter to the fixed point specified by the `viewCenter` parameter. That point is constrained by the current field of view of the movie. The values you pass in the `QTVRFloatPoint` structure are adjusted so that the magnified area does not show anything outside the view.

If the `kQTVRTranslation` control setting is disabled, the `QTVRSetViewCenter` function returns the result code `constraintReachedErr` and doesn't change the current view center.

SPECIAL CONSIDERATIONS

`QTVRSetViewCenter` is valid only for object nodes.

SEE ALSO

Use `QTVRGetViewCenter` (page 96) to get the view center of a movie. Use `QTVRSetControlSetting` (page 140) to control the setting of `kQTVRTranslation`.

QTVRNudge

You can use the `QTVRNudge` function to turn one step in a particular direction and display the new view.

```
OSErr QTVRNudge (QTVRInstance qtvr, QTVRNudgeControl direction);
```

`qtvr`	An instance of a QuickTime VR movie.
`direction`	The direction of the nudge. See "Nudge Directions" (page 81) for a description of the values you can pass in this parameter.
function result	A result code.

DESCRIPTION

The `QTVRNudge` function adjusts the current view of the movie specified by the `qtvr` parameter as indicated by the `direction` parameter. In particular, `QTVRNudge` turns one step in the indicated direction and displays the new view.

For example, to move to the next view that is right and up from the current view, set the `direction` parameter to `kQTVRUpRight` (that is, $\pi/4$ radians, or 45 degrees). Any value of the `direction` parameter that is not predefined is mapped to the closest defined value. For objects, if no view is located at the adjacent object view defined by the nudge direction and wrapping is off in the desired direction, then `QTVRNudge` remains at the current view and returns the result code `constraintReachedErr`.

For objects, `QTVRNudge` is useful for changing to an adjacent view without having to know the new pan and tilt angles.

The direction of the nudge is affected by the current control settings. See "Control Settings" (page 74) for more information.

SEE ALSO

Use `QTVRSetPanAngle` (page 92) and `QTVRSetTiltAngle` (page 94) to move to a new view specified using pan and tilt angles.

QTVRInteractionNudge

You can use the `QTVRInteractionNudge` function either to translate the image and display the new view or to rotate the object in a particular direction and display its new appearance. The current setting of the interaction property for `kQTVRInteractionNudgeMode` determines whether the nudge action translates the image or rotates the object.

```
OSErr QTVRInteractionNudge (QTVRInstance qtvr, QTVRNudgeControl
                        direction);
```

`qtvr` An instance of a QuickTime VR movie.

`direction` The direction of the nudge. See "Nudge Directions" (page 81) for a description of the values you can pass in this parameter. See "Nudge Mode" (page 80) for a description of the way you set the nudge mode.

function result A result code.

DESCRIPTION

The `QTVRInteractionNudge` function adjusts the current view of the movie specified by the `qtvr` parameter as indicated by the `direction` parameter. The type of adjustment depends on the property setting for nudge interaction mode. If the nudge interaction mode is `kQTVRNudgeRotate`, the action of the `QTVRInteractionNudge` function is to rotate the object in the specified direction. If the nudge interaction mode is `kQTVRNudgeTranslate`, the action of the `QTVRInteractionNudge` function is to translate the image in the specified direction.

If the nudge interaction mode is `kQTVRUNudgeSameAsMouse`, the action of the `QTVRInteractionNudge` function is determined by the current mouse mode.

SPECIAL CONSIDERATIONS

The `QTVRInteractionNudge` function is valid only for object nodes.

SEE ALSO

Use the functions `QTVRGetInteractionProperty` (page 163) and `QTVRSetInteractionProperty` (page 164) to set the nudge mode and direction properties.

QTVRShowDefaultView

You can use the `QTVRShowDefaultView` function to display the default view of a node.

```
OSErr QTVRShowDefaultView (QTVRInstance qtvr);
```

`qtvr` An instance of a QuickTime VR movie.

function result A result code.

DISCUSSION

The `QTVRShowDefaultView` function sets the default values of the pan angle, tilt angle, field of view, view center (for object nodes), default state, mouse-down

state, and all applicable animation and control settings for the node specified by the `qtvr` parameter. A node's default values are stored in the movie file.

Getting Scene and Node Information

The QuickTime VR Manager provides functions that you can use to get the VR world scene description atom container and to get and set a scene's current node.

QTVRGetVRWorld

You can use the `QTVRGetVRWorld` function to get the VR world atom container for a movie.

```
OSErr QTVRGetVRWorld (QTVRInstance qtvr, QTAtomContainer *vrWorld);
```

`qtvr`	An instance of a QuickTime VR movie.
`vrWorld`	On exit, a pointer to an atom container that contains information about the specified movie.
function result	A result code.

DESCRIPTION

The `QTVRGetVRWorld` function returns, in the `vrWorld` parameter, a pointer to an atom container that contains general scene information about the QuickTime VR movie specified by the `qtvr` parameter, as well as a list of all the nodes in that movie. You can use the QuickTime atom functions (introduced in QuickTime version 2.1) to extract atoms from that container.

Note
See the chapter "QuickTime VR Atom Containers" in this book for a description of the format of a VR world atom container and its associated atoms for a QuickTime VR movie. ◆

SPECIAL CONSIDERATIONS

The VR world atom container returned by `QTVRGetVRWorld` is a copy of the atom container maintained internally by the QuickTime VR Manager. You should dispose of the VR world atom container (by calling the QuickTime function `QTDisposeAtomContainer`) when you're finished using it.

QTVRGoToNodeID

You can use the `QTVRGoToNodeID` function to set the current node of a movie.

```
OSErr QTVRGoToNodeID (QTVRInstance qtvr, UInt32 nodeID);
```

`qtvr`	An instance of a QuickTime VR movie.
`nodeID`	The ID of the node you want to be the current node.
function result	A result code.

DESCRIPTION

The `QTVRGoToNodeID` function sets the current node in the QuickTime VR movie specified by the `qtvr` parameter to be the node that has the ID specified by the `nodeID` parameter.

The QuickTime VR Manager defines several constants for specific nodes. For example, you can set `nodeID` to `kQTVRDefaultNode` to set the current node to the default node in the scene. Similarly, you can set `nodeID` to `kQTVRPreviousNode` to return to the previous node. See "Node IDs" (page 58) for a description of the available node ID constants.

SPECIAL CONSIDERATIONS

Setting the current node also sets the pan, tilt, and field of view of the new current node to their default values. As a result, if you wish to set non-default angles, you should call `QTVRGoToNodeID` before you call `QTVRSetPanAngle`, `QTVRSetTiltAngle`, or `QTVRSetFieldOfView`.

SEE ALSO

Use `QTVRGetCurrentNodeID` (next) to get the current node ID.

QTVRGetCurrentNodeID

You can use the `QTVRGetCurrentNodeID` function to get the current node of a movie.

```
UInt32 QTVRGetCurrentNodeID (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	The ID of the current node of the specified movie.

DESCRIPTION

The `QTVRGetCurrentNodeID` function returns, as its function result, the ID of the current node of the QuickTime VR movie specified by the `qtvr` parameter.

SEE ALSO

Use `QTVRGoToNodeID` (page 102) to set the current node ID.

QTVRGetNodeType

You can use the `QTVRGetNodeType` function to get the type of a movie node.

```
OSType QTVRGetNodeType (QTVRInstance qtvr, UInt32 nodeID);
```

`qtvr`	An instance of a QuickTime VR movie.
`nodeID`	A node ID. Pass `kQTVRCurrentNode` for the current node.
function result	The type of the specified node.

DESCRIPTION

The `QTVRGetNodeType` function returns, as its function result, the type of the node specified by the `nodeID` parameter in the QuickTime VR movie specified by the `qtvr` parameter. See “Node Types” (page 58) for a description of the values that `QTVRGetNodeType` can return.

QTVRGetNodeInfo

You can use the `QTVRGetNodeInfo` function to get the node information atom container that describes a node and all the hot spots in the node.

```
OSErr QTVRGetNodeInfo (
                    QTVRInstance qtvr,
                    UInt32 nodeID,
                    QTAtomContainer *nodeInfo);
```

`qtvr`	An instance of a QuickTime VR movie.
`nodeID`	A node ID. Set this parameter to `kQTVRCurrentNode` to receive information about the current node.
`nodeInfo`	On exit, a pointer to an atom container that contains information about the specified node.
function result	A result code.

DESCRIPTION

The `QTVRGetNodeInfo` function returns, in the `nodeInfo` parameter, a pointer to an atom container that contains information about the node specified by the `nodeID` parameter in the movie specified by the `qtvr` parameter. The atom container includes information about all the hot spots contained in that node. You can use the QuickTime atom functions (introduced in QuickTime version 2.1) to extract atoms from that container. You can also use those functions to access the hot spot atom list. All hot spot atoms are contained in the hot spot parent atom.

Note
See the chapter "QuickTime VR Atom Containers" in this book for a description of the format of a node information atom container and its associated atoms for a node. ◆

SPECIAL CONSIDERATIONS

The node information atom container returned by `QTVRGetNodeInfo` is a copy of the atom container maintained internally by the QuickTime VR Manager. You should dispose of the node information atom container (by calling the QuickTime function `QTDisposeAtomContainer`) when you're finished using it.

SEE ALSO

Listing 4-1 (page 256) illustrates how to call the `QTVRGetNodeInfo` function.

Managing Hot Spots

The QuickTime VR Manager provides functions that you can use to manage the hot spots of a node.

QTVRPtToHotSpotID

You can use the `QTVRPtToHotSpotID` function to get the ID of the hot spot, if any, that lies beneath a point.

```
OSErr QTVRPtToHotSpotID (QTVRInstance qtvr, Point pt, UInt32 *hotSpotID);
```

`qtvr`	An instance of a QuickTime VR movie.
`pt`	A point, in the local coordinates of the graphics world of the specified movie.
`hotSpotID`	On entry, a pointer to a long integer. On exit, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.
function result	A result code.

DESCRIPTION

The `QTVRPtToHotSpotID` function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0.

QTVRGetHotSpotType

You can use the `QTVRGetHotSpotType` function to get the type of a hot spot.

```
OSErr QTVRGetHotSpotType (
                    QTVRInstance qtvr,
                    UInt32 hotSpotID,
                    OSType *hotSpotType);
```

`qtvr`	An instance of a QuickTime VR movie.
`hotSpotID`	A hot spot ID.
`hotSpotType`	On entry, a pointer to a long integer. On exit, that long integer contains the type of the hot spot specified by the hot spot ID.
function result	A result code.

DESCRIPTION

The `QTVRGetHotSpotType` function gets the type of a hot spot whose ID you specify. In combination with the `kQTVRGetHotSpotTypeSelector` intercept selector, this allows an application to change a hot spot's type dynamically.

For example, an application can take an existing movie and cause VR to display the cursors for a type of hotspot different from the one the movie was originally authored with. In combination with intercepting `kQTVRTriggerHotSpotSelector`, this would allow an Internet plugin to override undefined or link hotspots in movies to make them appear and act as though they are URL links instead. If `kQTVRTriggerHotSpotSelector` is not intercepted, VR will attempt to act on the hotspot in the normal way; by storing both link and URL data in a file, the exact behavior can be determined at runtime by an application to allow linking to either another node locally or a remote URL link, depending on system configuration or other arbitrary considerations.

SEE ALSO

Use `kQTVRGetHotSpotTypeSelector` to set the selector for the intercept procedure.

QTVRTriggerHotSpot

You can use the `QTVRTriggerHotSpot` function to trigger a hot spot.

```
OSErr QTVRTriggerHotSpot (
                    QTVRInstance qtvr,
                    UInt32 hotSpotID,
                    QTAtomContainer nodeInfo,
                    QTAtom selectedAtom);
```

`qtvr`	An instance of a QuickTime VR movie.
`hotSpotID`	A hot spot ID.
`nodeInfo`	A node information atom container (obtained from a previous call to `QTVRGetNodeInfo`). You can pass the value 0 in this parameter to have the QuickTime VR Manager determine the appropriate node information atom container.
`selectedAtom`	The atom of the hot spot to trigger. You can pass the value 0 in this parameter to have the QuickTime VR Manager determine the appropriate hot spot atom.
function result	A result code.

DESCRIPTION

One way you can use the `QTVRTriggerHotSpot` function is to execute any hot spot without the user's having clicked the hot spot. Usually, you need only specify the `qtvr` instance and the hot spot ID. You can pass zero for the `nodeInfo` and `selectedAtom` parameters.

The more common use of the `QTVRTriggerHotSpot` function is not in calling it directly, but in setting up an intercept procedure on it. The `QTVRTriggerHotSpot` function is called internally by the QuickTime VR Manager whenever a user clicks a hot spot. You can intercept calls to trigger your custom hot spots, which allows you to perform any custom actions you desire.

When the `QTVRTriggerHotSpot` function is called internally (and then intercepted by your intercept procedure), the `nodeInfo` and `selectedAtom` parameters have been properly set by the QuickTime VR Manager and are available for your use. For undefined hot spots that do not have an associated hot spot atom in the node info atom container, the `selectedAtom` parameter will be set to zero.

SPECIAL CONSIDERATIONS

You can call `QTVRTriggerHotSpot` even on hot spots that are currently disabled.

SEE ALSO

Use `QTVRInstallInterceptProc` (page 125) to install an intercept procedure. Listing 4-2 in Chapter 4, "QuickTime VR Atom Containers," shows an example of using an intercept procedure on the `QTVRTriggerHotSpot` function.

QTVREnableHotSpot

You can use the `QTVREnableHotSpot` function to enable or disable one or more hot spots.

```
OSErr QTVREnableHotSpot (
                    QTVRInstance qtvr,
                    UInt32 enableFlag,
                    UInt32 hotSpotValue,
                    Boolean enable);
```

`qtvr`	An instance of a QuickTime VR movie.
`enableFlag`	The kind of hot spot or hot spots to enable or disable. See "Hot Spot Selectors" (page 72) for a description of the constants you can pass in this parameter.
`hotSpotValue`	The desired hot spot or spots, relative to the specified enabled flag. If `enableFlag` is `kQTVRHotSpotID`, this parameter specifies a hot spot ID. If `enableFlag` is `kQTVRHotSpotType`, this parameter specifies a hot spot type. If `enableFlag` is `kQTVRAllHotSpots`, this parameter should be set to zero.
`enable`	A Boolean value that indicates whether the specified hot spots are to be enabled (`true`) or disabled (`false`).
function result	A result code.

DESCRIPTION

The `QTVREnableHotSpot` function either enables or disables the hot spot or spots specified by the `enableFlag` and `hotSpotValue` parameters, according to the value of the `enable` parameter. The hot spots are always selected from among the hot spots in the current node of the QuickTime VR movie specified by the `qtvr` parameter.

Normally, all hot spots in a node are enabled (that is, the cursor automatically changes shape when it is moved over a hot spot, and the `QTVRTriggerHotSpot` function is called internally when the user clicks a hot spot). When a hot spot is disabled, QuickTime VR behaves as if the hot spot were not present.

QTVRSetMouseOverHotSpotProc

You can use the `QTVRSetMouseOverHotSpotProc` function to install or remove a mouse over hot spot procedure.

```
OSErr
QTVRSetMouseOverHotSpotProc (
                    QTVRInstance qtvr,
                     QTVRMouseOverHotSpotUPP  mouseOverHotSpotProc,
                     SInt32         refCon,
                     UInt32         flags);
```

qtvr — An instance of a QuickTime VR movie.

`mouseOverHotSpotProc` — A universal procedure pointer for a mouse over hot spot procedure. See “Mouse Over Hot Spot Procedure” (page 180) for information about mouse over hot spot procedures.

`refCon` — A reference constant. This value is passed to the specified mouse over hot spot procedure.

`flags` — Unused. Set this parameter to 0.

function result — A result code.

DESCRIPTION

The `QTVRSetMouseOverHotSpotProc` function installs the routine specified by the `mouseOverHotSpotProc` parameter as a mouse over hot spot procedure for the QuickTime VR movie specified by the `qtvr` parameter. Subsequent user actions (such as moving the cursor over an enabled hot spot in that movie) cause the callback routine to be executed. The reference constant specified by the `refCon` parameter is passed unchanged to your callback routine.

To remove a previously installed callback routine, set `mouseOverHotSpotProc` to `nil`.

IMPORTANT

Your mouse over hot spot procedure is called only for enabled hot spots. ▲

QTVRGetVisibleHotSpots

You can use the `QTVRGetVisibleHotSpots` function to get a list of the currently visible hot spots in a QuickTime VR movie.

```
UInt32 QTVRGetVisibleHotSpots (QTVRInstance qtvr, Handle hotSpots);
```

`qtvr`	An instance of a QuickTime VR movie.
`hotSpots`	On entry, a handle to a block of memory. On exit, that block of memory is filled with a list of the IDs of the visible hot spots in the specified QuickTime VR movie. If necessary, the handle is resized to hold all the hot spot IDs. Accordingly, the handle must be unlocked at the time you call `QTVRGetVisibleHotSpots`.
function result	The number of hot spots whose IDs are returned though the `hotSpots` parameter.

DESCRIPTION

The `QTVRGetVisibleHotSpots` function returns, in the block of memory specified by the `hotSpots` parameter, a list of IDs of all the hot spots in the QuickTime VR movie specified by the `qtvr` parameter that are currently visible. `QTVRGetVisibleHotSpots` also returns, as its function result, the number of hot spot IDs returned though the `hotSpots` parameter.

The handle specified by the `hotSpots` parameter must be a valid handle created by the Memory Manager.

QTVRGetHotSpotRegion

You can use the `QTVRGetHotSpotRegion` function to get the region occupied by a hot spot.

```
OSErr QTVRGetHotSpotRegion (
                    QTVRInstance qtvr,
                    UInt32 hotSpotID,
                    RgnHandle hotSpotRegion);
```

`qtvr` An instance of a QuickTime VR movie.

`hotSpotID` A hot spot ID.

`hotSpotRegion` On entry, an initialized handle to a region (obtained by calling `NewRgn`). On exit, this region is rewritten with the region occupied by the hot spot having the specified ID.

function result A result code.

DESCRIPTION

The `QTVRGetHotSpotRegion` function returns, in the `hotSpotRegion` parameter, a handle to the region occupied by the hot spot, in the QuickTime VR movie specified by the `qtvr` parameter, whose ID is specified by the `hotSpotID` parameter. The region is clipped to the bounds of the movie's graphics world.

You can obtain the regions of all visible hot spots by calling the `QTVRGetVisibleHotSpots` function and then calling `QTVRGetHotSpotRegion` for each hot spot ID in the list.

SPECIAL CONSIDERATIONS

The first time you call `QTVRGetHotSpotRegion`, a significant amount of memory might need to be allocated. Accordingly, you should always check for Memory Manager errors returned by `QTVRGetHotSpotRegion`.

Your application is responsible for disposing of the memory occupied by the returned region.

Handling Events

The QuickTime VR Manager provides a number of routines that you can use to handle user actions such as moving the mouse or clicking the mouse button. Normally, QuickTime VR handles all user interaction internally if your application calls the `MCIsPlayerEvent` function in its main event loop. For code that does not have a main event loop (such as an OpenDoc part), you might need to override QuickTime VR's mouse event handling.

Note
Most applications do not need to use the functions described in this section. ◆

QTVRGetMouseOverTracking

You can use the `QTVRGetMouseOverTracking` function to get the current state of mouse-over tracking.

```
Boolean QTVRGetMouseOverTracking (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A Boolean value that indicates whether QuickTime VR is currently handling mouse-over tracking for the specified movie (`true`) or not (`false`).

DESCRIPTION

The `QTVRGetMouseOverTracking` function returns, as its function result, a Boolean value that indicates whether QuickTime VR is currently handling mouse-over tracking for the QuickTime VR movie specified by the `qtvr` parameter (`true`) or not (`false`). By default, QuickTime VR tracks mouse movements in a QuickTime VR movie and changes the shape of the cursor as appropriate.

SEE ALSO

Use `QTVRSetMouseOverTracking` (next) to change the mouse-over tracking state of a QuickTime VR movie.

QTVRSetMouseOverTracking

You can use the `QTVRSetMouseOverTracking` function to set the state of mouse-over tracking.

```
OSErr QTVRSetMouseOverTracking (QTVRInstance qtvr, Boolean enable);
```

`qtvr`	An instance of a QuickTime VR movie.
`enable`	A Boolean value that indicates whether QuickTime VR should handle mouse-over tracking for the specified movie (`true`) or not (`false`).
function result	A result code.

DESCRIPTION

The `QTVRSetMouseOverTracking` function sets the mouse-over tracking state of the QuickTime VR movie specified by the `qtvr` parameter to the state specified by the `enable` parameter. By default, QuickTime VR tracks mouse movements in a QuickTime VR movie and changes the shape of the cursor as appropriate. If you disable mouse-over tracking (by passing `false` in the `enable` parameter), you must call the `QTVRMouseEnter`, `QTVRMouseWithin`, and `QTVRMouseLeave` functions at the appropriate times to handle user actions.

SEE ALSO

Use `QTVRGetMouseOverTracking` (page 112) to get the current mouse-over tracking state of a QuickTime VR movie.

QTVRMouseEnter

You can use the `QTVRMouseEnter` function to handle the user's moving the cursor into a QuickTime VR movie for which mouse-over tracking is disabled.

```
OSErr QTVRMouseEnter (
                    QTVRInstance qtvr,
                    Point pt,
                    UInt32 *hotSpotID,
                    WindowPtr w);
```

`qtvr`	An instance of a QuickTime VR movie.
`pt`	The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.
`hotSpotID`	On entry, a pointer to a long integer. On exit, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.
`w`	A pointer to a graphics world.
function result	A result code.

DESCRIPTION

The `QTVRMouseEnter` function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseEnter` also performs any other tasks that are typically performed when the user first moves the cursor into a QuickTime VR movie.

SPECIAL CONSIDERATIONS

You need to call `QTVRMouseEnter` only if you have disabled mouse-over tracking for the specified QuickTime VR movie.

SEE ALSO

Use `QTVRSetMouseOverTracking` (page 113) to change the mouse-over tracking state of a QuickTime VR movie. Use `QTVRMouseWithin` (next) and

QTVRMouseLeave (page 116) to handle the cursor's remaining in and leaving a QuickTime VR movie.

QTVRMouseWithin

You can use the QTVRMouseWithin function to handle the user's leaving the cursor in a QuickTime VR movie for which mouse-over tracking is disabled.

```
OSErr QTVRMouseWithin (QTVRInstance qtvr, Point pt, UInt32 *hotSpotID,
                       WindowPtr w);
```

qtvr	An instance of a QuickTime VR movie.
pt	The current location of the cursor, in the local coordinates of the graphics world specified by the w parameter.
hotSpotID	On entry, a pointer to a long integer. On exit, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.
w	A pointer to a graphics world.
function result	A result code.

DESCRIPTION

The QTVRMouseWithin function returns, in the long integer pointed to by the hotSpotID parameter, the ID of the hot spot in the QuickTime VR movie specified by the qtvr parameter that lies directly under the point specified by the pt parameter. If no hot spot lies under that point, the long integer is set to 0. QTVRMouseWithin also performs any other tasks that are typically performed when the user leaves the cursor in a QuickTime VR movie.

You should call QTVRMouseWithin repeatedly for as long as the cursor remains in the specified QuickTime VR movie.

SPECIAL CONSIDERATIONS

You need to call QTVRMouseWithin only if you have disabled mouse-over tracking for the specified QuickTime VR movie.

SEE ALSO

Use `QTVRSetMouseOverTracking` (page 113) to change the mouse-over tracking state of a QuickTime VR movie. Use `QTVRMouseEnter` (page 114) and `QTVRMouseLeave` (next) to handle the cursor's entering and leaving a QuickTime VR movie.

QTVRMouseLeave

You can use the `QTVRMouseLeave` function to handle the user's moving the cursor out of a QuickTime VR movie for which mouse-over tracking is disabled.

```
OSErr QTVRMouseLeave (QTVRInstance qtvr, Point pt, WindowPtr w);
```

`qtvr`	An instance of a QuickTime VR movie.
`pt`	The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.
`w`	A pointer to a graphics world.
function result	A result code.

DESCRIPTION

The `QTVRMouseLeave` function performs any tasks that are typically performed when the user moves the cursor out of a QuickTime VR movie.

SPECIAL CONSIDERATIONS

You need to call `QTVRMouseLeave` only if you have disabled mouse-over tracking for the specified QuickTime VR movie.

SEE ALSO

Use `QTVRSetMouseOverTracking` (page 113) to change the mouse-over tracking state of a QuickTime VR movie. Use `QTVRMouseEnter` (page 114) and `QTVRMouseWithin` (page 115) to handle the cursor's entering and remaining in a QuickTime VR movie.

QTVRGetMouseDownTracking

You can use the `QTVRGetMouseDownTracking` function to get the current state of mouse-down tracking.

```
Boolean QTVRGetMouseDownTracking (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A Boolean value that indicates whether QuickTime VR is currently handling mouse-down tracking for the specified movie (`true`) or not (`false`).

DESCRIPTION

The `QTVRGetMouseDownTracking` function returns, as its function result, a Boolean value that indicates whether QuickTime VR is currently handling mouse-down tracking for the QuickTime VR movie specified by the `qtvr` parameter (`true`) or not (`false`). By default, QuickTime VR tracks mouse clicks in a QuickTime VR movie and triggers hot spots as necessary.

SEE ALSO

Use `QTVRSetMouseDownTracking` (next) to change the mouse-down tracking state of a QuickTime VR movie.

QTVRSetMouseDownTracking

You can use the `QTVRSetMouseDownTracking` function to set the state of mouse-down tracking.

```
OSErr QTVRSetMouseDownTracking (QTVRInstance qtvr, Boolean enable);
```

`qtvr`	An instance of a QuickTime VR movie.
`enable`	A Boolean value that indicates whether QuickTime VR should handle mouse-down tracking for the specified movie (`true`) or not (`false`).

function result A result code.

DESCRIPTION

The `QTVRSetMouseDownTracking` function sets the mouse-down tracking state of the QuickTime VR movie specified by the `qtvr` parameter to the state specified by the `enable` parameter. By default, QuickTime VR tracks mouse clicks in a QuickTime VR movie and triggers hot spots as appropriate. If you disable mouse-down tracking (by passing `false` in the `enable` parameter), you must call the `QTVRMouseDown`, `QTVRMouseStillDown`, and `QTVRMouseUp` functions at the appropriate times to handle user actions.

SEE ALSO

Use `QTVRGetMouseDownTracking` (page 117) to get the current mouse-down tracking state of a QuickTime VR movie.

QTVRMouseDown

You can use the `QTVRMouseDown` function to handle the user's clicking the mouse button when the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseDown (QTVRInstance qtvr,
                     Point pt,
                     UInt32 when,
                     UInt16 modifiers,
                     UInt32 *hotSpotID,
                     WindowPtr w);
```

`qtvr` An instance of a QuickTime VR movie.

`pt` The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

`when` The time, in the number of ticks since system startup, when the mouse-down event was posted.

`modifiers`	A short integer that contains information about the state of the modifier keys and the mouse button at the time the event was posted. See the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for a description of the information encoded in this parameter.
`hotSpotID`	On entry, a pointer to a long integer. On exit, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.
`w`	A pointer to a graphics world.
function result	A result code.

DESCRIPTION

The `QTVRMouseDown` function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseDown` also performs any other tasks that are typically performed when the user clicks the mouse button when the cursor is in a QuickTime VR movie.

SPECIAL CONSIDERATIONS

You need to call `QTVRMouseDown` only if you have disabled mouse-down tracking for the specified QuickTime VR movie.

SEE ALSO

Use `QTVRSetMouseDownTracking` (page 117) to change the mouse-down tracking state of a QuickTime VR movie. Use `QTVRMouseUp` (page 121) and `QTVRMouseStillDown` (next) to handle the mouse button's being held down and released.

QTVRMouseStillDown

You can use the `QTVRMouseStillDown` function to handle the user's holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

IMPORTANT

Applications running on operating systems other than Mac OS should use the extended form of this function: see "QTVRMouseStillDownExtended" (page 122). ▲

```
OSErr QTVRMouseStillDown (
                    QTVRInstance qtvr,
                    Point pt,
                    UInt32 *hotSpotID,
                    WindowPtr w);
```

`qtvr`	An instance of a QuickTime VR movie.
`pt`	The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.
`hotSpotID`	On entry, a pointer to a long integer. On exit, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.
`w`	A pointer to a graphics world.
function result	A result code.

DESCRIPTION

The `QTVRMouseStillDown` function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseStillDown` also performs any other tasks that are typically performed when the user holds down the mouse button when the cursor is in a QuickTime VR movie.

You should call `QTVRMouseStillDown` repeatedly for as long as the user holds down the mouse button while the cursor is in the specified QuickTime VR movie.

SPECIAL CONSIDERATIONS

You need to call `QTVRMouseStillDown` only if you have disabled mouse-down tracking for the specified QuickTime VR movie.

SEE ALSO

Use `QTVRSetMouseDownTracking` (page 117) to change the mouse-down tracking state of a QuickTime VR movie. Use `QTVRMouseDown` (page 118) and `QTVRMouseUp` (next) to handle the mouse button's being clicked and released.

QTVRMouseUp

You can use the `QTVRMouseUp` function to handle the user's releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

IMPORTANT

Applications running on operating systems other than Mac OS should use the extended form of this function: see "QTVRMouseUpExtended" (page 124). ▲

```
OSErr QTVRMouseUp (QTVRInstance qtvr, Point pt, UInt32 *hotSpotID,
                   WindowPtr w);
```

`qtvr`	An instance of a QuickTime VR movie.
`pt`	The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.
`hotSpotID`	On entry, a pointer to a long integer. On exit, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.
`w`	A pointer to a graphics world.
function result	A result code.

DESCRIPTION

The `QTVRMouseUp` function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseUp` also performs any other tasks that are typically performed when the user releases the mouse button after clicking it when the cursor is in a QuickTime VR movie.

SPECIAL CONSIDERATIONS

You need to call `QTVRMouseUp` only if you have disabled mouse-down tracking for the specified QuickTime VR movie.

SEE ALSO

Use `QTVRSetMouseDownTracking` (page 117) to change the mouse-down tracking state of a QuickTime VR movie. Use `QTVRMouseDown` (page 118) and `QTVRMouseStillDown` (page 119) to handle the mouse button's being clicked and held down.

QTVRMouseStillDownExtended

You can use the `QTVRMouseStillDownExtended` function the same way you use the `QTVRMouseStillDown` function, to handle the user's holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

The `QTVRMouseStillDownExtended` function uses the same intercept as the `QTVRMouseStillDown` function but has two additional parameters. Applications that intercept `kQTVRMouseStillDownExtended` should always check the `paramCount` field to make sure it is 5 before using the last two fields.

Internally, QuickTime VR always uses the `QTVRMouseStillDownExtended` function instead of `QTVRMouseStillDown`. Developers implementing their own mouse down tracking don't need to use the extended version unless they also intercept the procedure and need the added parameters.

```
OSErr QTVRMouseStillDownExtended (
                    QTVRInstance qtvr,
                    Point pt,
                    UInt32 *hotSpotID,
                    WindowPtr w,
                    UInt32 when,
                    UInt16 modifiers);
```

`qtvr` An instance of a QuickTime VR movie.

`pt` The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

`hotSpotID`	On entry, a pointer to a long integer. On exit, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.
`w`	A pointer to a graphics world.
`when`	The current time, in the number of ticks since system startup, obtained by a call to `TickCount()`.
`modifiers`	A short integer that contains information about the state of the modifier keys and the mouse button at the current time. See the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for a description of the information encoded in this parameter.
function result	A result code.

DESCRIPTION

The `QTVRMouseStillDownExtended` function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseStillDownExtended` also performs any other tasks that are typically performed when the user holds down the mouse button when the cursor is in a QuickTime VR movie.

You should call `QTVRMouseStillDownExtended` repeatedly for as long as the user holds down the mouse button while the cursor is in the specified QuickTime VR movie.

SPECIAL CONSIDERATIONS

You need to call `QTVRMouseStillDownExtended` only if you have disabled mouse-down tracking for the specified QuickTime VR movie.

SEE ALSO

Use `QTVRSetMouseDownTracking` (page 117) to change the mouse-down tracking state of a QuickTime VR movie. Use `QTVRMouseDown` (page 118) and `QTVRMouseUpExtended` (next) to handle the mouse button's being clicked and released.

QTVRMouseUpExtended

You can use the `QTVRMouseUpExtended` function the same way you use the `QTVRMouseUp` function, to handle the user's releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

The `QTVRMouseUpExtended` function uses the same intercept as the `QTVRMouseUp` function but has two additional parameters. Applications that intercept `QTVRMouseUpExtended` should always check the `paramCount` field to make sure it is 5 before using the last two fields.

Internally, QuickTime VR always uses the `QTVRMouseUpExtended` function instead of `QTVRMouseUp`. Developers implementing their own mouse down tracking don't need to use the extended version unless they also intercept the procedure and need the added parameters.

```
OSErr QTVRMouseUpExtended (
                    QTVRInstance qtvr,
                    Point pt,
                    UInt32 *hotSpotID,
                    WindowPtr w,
                    UInt32 when,
                    UInt16 modifiers);
```

`qtvr` An instance of a QuickTime VR movie.

`pt` The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

`hotSpotID` On entry, a pointer to a long integer. On exit, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

`w` A pointer to a graphics world.

`when` The time, in the number of ticks since system startup, when the mouse-up event was posted.

`modifiers` A short integer that contains information about the state of the modifier keys and the mouse button at the time the event was posted. See the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for a description of the information encoded in this parameter.

function result A result code.

DESCRIPTION

The `QTVRMouseUpExtended` function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseUpExtended` also performs any other tasks that are typically performed when the user releases the mouse button after clicking it when the cursor is in a QuickTime VR movie.

SPECIAL CONSIDERATIONS

You need to call `QTVRMouseUp` or `QTVRMouseUpExtended` only if you have disabled mouse-down tracking for the specified QuickTime VR movie.

Intercepting QuickTime VR Manager Routines

The QuickTime VR Manager provides a function that you can use to install an intercept procedure. Your intercept procedure can provide either alternative or additional functionality to that provided by the intercepted procedure.

QTVRInstallInterceptProc

You can use the `QTVRInstallInterceptProc` function to install or remove an intercept procedure for a QuickTime VR Manager function.

```
OSErr QTVRInstallInterceptProc (
                    QTVRInstance qtvr,
                    QTVRProcSelector selector,
                    QTVRInterceptUPP interceptProc,
                    SInt32 refCon,
                    UInt32 flags);
```

`qtvr` An instance of a QuickTime VR movie.

`selector` A selector that indicates which QuickTime VR function to intercept. See “Intercept Selectors” (page 60) for a description of the available intercept selectors.

`interceptProc`	A universal procedure pointer for an intercept procedure. See "QuickTime VR Intercept Procedure" (page 181) for information about intercept procedures. Set this parameter to `nil` to remove a previously installed intercept procedure.
`refCon`	A reference constant. This value is passed to the specified intercept routine.
`flags`	Unused. Set this parameter to 0.
function result	A result code.

DESCRIPTION

The `QTVRInstallInterceptProc` function installs the procedure specified by the `interceptProc` parameter as an intercept procedure for the QuickTime VR function specified by the `selector` parameter for the QuickTime VR movie specified by the `qtvr` parameter. Your intercept procedure is called whenever QuickTime VR is about to execute the function you are intercepting. Your procedure can simply replace the intercepted function (by returning the value true in its cancel parameter); it can call through to the intercepted function (by calling the `QTVRCallInterceptedProc` function); or it can allow the intercepted function to execute when the intercept procedure returns (by returning the value `false` in its `cancel` parameter).

SEE ALSO

For examples of the use of the `QTVRInstallInterceptProc` function, see Listing 2-7 (page 53) and Listing 4-2 in Chapter 4, "QuickTime VR Atom Containers."

QTVRCallInterceptedProc

You can use the `QTVRCallInterceptedProc` function to call an intercepted QuickTime VR function from within an intercept procedure.

```
OSErr QTVRCallInterceptedProc (
                    QTVRInstance qtvr,
                    QTVRInterceptRecord *qtvrMsg);
```

`qtvr`	An instance of a QuickTime VR movie.
`qtvrMsg`	An intercept record (page 85) that specifies the function that your procedure is intercepting and the parameters for that function. This should be the same intercept record passed to your intercept procedure.
function result	A result code.

DESCRIPTION

The `QTVRCallInterceptedProc` function executes the QuickTime VR Manager function indicated by the `selector` field of the `qtvrMsg` intercept record. The parameters passed to that function are the QuickTime VR movie specified by the `qtvr` parameter and any other parameters contained in the `parameter` field of the `qtvrMsg` record. You can, if you wish, change the parameters in that field before calling `QTVRCallInterceptedProc`.

You can call `QTVRCallInterceptedProc` more than once in your intercept procedure. In addition, the QuickTime VR Manager will call the intercepted function again unless your intercept procedure returns `true` in its `cancel` parameter.

SPECIAL CONSIDERATIONS

You should call `QTVRCallInterceptedProc` only in an intercept procedure.

SEE ALSO

See "QuickTime VR Intercept Procedure" (page 181) for information about intercept procedures. Listing 2-6 (page 51) illustrates the use of `QTVRCallInterceptedProc`.

Managing Object Nodes

The QuickTime VR Manager provides functions that you can use to manage object nodes.

QTVRGetCurrentMouseMode

You can use the `QTVRGetCurrentMouseMode` function to obtain the current mouse control modes.

The value returned by the `QTVRGetCurrentMouseMode` function is an unsigned long integer that encodes the current mouse control modes. If a bit in the integer is set, the corresponding mode is one of the current mouse modes. The mode bits are addressed using these constants:

```
enum {
    kQTVRPanning                                = 1L << 0,
    kQTVRTranslating                            = 1L << 1,
    kQTVRZooming                                = 1L << 2,
    kQTVRScrolling                              = 1L << 3,
    kQTVRSelecting                              = 1L << 4
};
```

Notice that several modes can be returned. That means a return value could have both zooming and translating set, for example.

Constant descriptions

`kQTVRPanning`	If this bit is set, the mouse controls panning of standard objects, using objects-only controllers.
`kQTVRTranslating`	If this bit is set, the mouse controls translation for all objects.
`kQTVRZooming`	If this bit is set, the mouse controls zooming for all objects.
`kQTVRScrolling`	If this bit is set, the mouse controls arrow scrolling for standard objects and scrolling for joystick objects.
`kQTVRSelecting`	If this bit is set, the mouse controls selecting of objects as an object absolute controller.

QTVRGetFrameRate

You can use the `QTVRGetFrameRate` function to get the current frame rate of an object node.

```
float QTVRGetFrameRate (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A floating-point value that represents the current frame rate of the specified movie.

DESCRIPTION

The `QTVRGetFrameRate` function returns, as its function result, the current frame rate of the object node specified by the `qtvr` parameter. A frame rate is a floating-point value in the range from –100.0 to +100.0. An object node's default frame rate is stored in the movie file.

SPECIAL CONSIDERATIONS

`QTVRGetFrameRate` is valid only for object nodes.

SEE ALSO

Use `QTVRSetFrameRate` (next) to set the frame rate of an object node.

QTVRSetFrameRate

You can use the `QTVRSetFrameRate` function to set the frame rate of an object node.

```
OSErr QTVRSetFrameRate (QTVRInstance qtvr, float rate);
```

`qtvr`	An instance of a QuickTime VR movie.

`rate`	The desired frame rate of the specified movie. A frame rate is a floating-point value in the range from –100.0 to +100.0. Positive values indicate forward rates, and negative values indicate reverse rates. Set this parameter to 0 to stop the movie.
function result	A result code.

DESCRIPTION

The `QTVRSetFrameRate` function sets the frame rate of the object node specified by the `qtvr` parameter to the rate specified by the `rate` parameter. This function is most useful when an object is being viewed with a looping animation. (The current view of the object may contain frames that are played in a loop, as specified by the file format.) You can use `QTVRSetFrameRate` to change the frame rate of the loop.

If the value specified in the `rate` parameter lies outside the valid range, `QTVRSetFrameRate` returns the result code `constraintReachedErr` and sets the frame rate to the nearest constraint.

SPECIAL CONSIDERATIONS

`QTVRSetFrameRate` is valid only for object nodes.

SEE ALSO

Use `QTVRGetFrameRate` (page 129) to get the frame rate of an object node.

QTVRGetViewRate

You can use the `QTVRGetViewRate` function to get the current view rate of an object node.

```
float QTVRGetViewRate (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A floating-point value that represents the current view rate of the specified movie.

DESCRIPTION

The `QTVRGetViewRate` function returns, as its function result, the current view rate of the object node specified by the `qtvr` parameter. A view rate is a floating-point value in the range from –100.0 to +100.0. An object node's default view rate is stored in the movie file.

SPECIAL CONSIDERATIONS

`QTVRGetViewRate` is valid only for object nodes.

SEE ALSO

Use `QTVRSetViewRate` (next) to set the view rate of an object node.

QTVRSetViewRate

You can use the `QTVRSetViewRate` function to set the view rate of an object node.

```
OSErr QTVRSetViewRate (QTVRInstance qtvr, float rate);
```

`qtvr`	An instance of a QuickTime VR movie.
`rate`	The desired view rate of the specified movie. A view rate is a floating-point value in the range from –100.0 to +100.0. Positive values indicate forward rates, and negative values indicate reverse rates. Set this parameter to 0 to stop the movie.
function result	A result code.

DESCRIPTION

The `QTVRSetViewRate` function sets the view rate of the object node specified by the `qtvr` parameter to the rate specified by the `rate` parameter. A node's view rate might be modified by the current animation settings.

If the value specified in the `rate` parameter lies outside the valid range, `QTVRSetViewRate` returns the result code `constraintReachedErr` and sets the view rate to the nearest constraint.

SPECIAL CONSIDERATIONS

`QTVRSetViewRate` is valid only for object nodes.

SEE ALSO

Use `QTVRGetViewRate` (page 130) to get the current view rate of an object node.

QTVRGetCurrentViewDuration

You can use the `QTVRGetCurrentViewDuration` function to get the duration of the current view of an object node.

```
TimeValue QTVRGetCurrentViewDuration (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	The interval of time from the beginning to the end of the current view.

DESCRIPTION

The `QTVRGetCurrentViewDuration` function returns, as its function result, the duration of the current view of the object node specified by the `qtvr` parameter.

SPECIAL CONSIDERATIONS

`QTVRGetCurrentViewDuration` is valid only for object nodes.

You cannot change a node's view duration.

QTVRGetViewCurrentTime

You can use the `QTVRGetViewCurrentTime` function to get the current time in the current view.

```
TimeValue QTVRGetViewCurrentTime (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	The current time in the current view.

DESCRIPTION

The `QTVRGetViewCurrentTime` function returns, as its function result, the current time in the current view of the object node specified by the `qtvr` parameter. The returned value is always greater than or equal to 0 and less than or equal to the value returned by the `QTVRGetCurrentViewDuration` function.

SPECIAL CONSIDERATIONS

`QTVRGetViewCurrentTime` is valid only for object nodes.

SEE ALSO

Use `QTVRSetViewCurrentTime` (next) to set the current time of an object node.

QTVRSetViewCurrentTime

You can use the `QTVRSetViewCurrentTime` function to set the time in the current view.

```
OSErr QTVRSetViewCurrentTime (QTVRInstance qtvr, TimeValue time);
```

`qtvr`	An instance of a QuickTime VR movie.
`time`	The desired time in the current view.
function result	A result code.

DESCRIPTION

The `QTVRSetViewCurrentTime` function sets the current time in the current view of the object node specified by the `qtvr` parameter to the value specified by the `time` parameter. That value should be greater than or equal to 0 and less than or equal to the value returned by the `QTVRGetCurrentViewDuration` function.

`QTVRSetViewCurrentTime` returns the result code `timeNotInViewErr` if the specified time value is greater than or equal to the view duration of the specified object node; in addition, `QTVRSetViewCurrentTime` sets the current view time to 1 less than the view duration. Similarly, `QTVRSetViewCurrentTime` returns the result code `timeNotInViewErr` if the specified time value is less than 0; in that case, `QTVRSetViewCurrentTime` sets the current view time to 0.

SPECIAL CONSIDERATIONS

`QTVRSetViewCurrentTime` is valid only for object nodes.

SEE ALSO

Use `QTVRGetViewCurrentTime` (page 133) to get the current time of an object node.

QTVRGetViewStateCount

You can use the `QTVRGetViewStateCount` function to get the number of view states of an object node.

```
UInt16 QTVRGetViewStateCount (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	The number of view states associated with the specified object node.

DESCRIPTION

The `QTVRGetViewStateCount` function returns, as its function result, the number of view states associated with the object node specified by the `qtvr` parameter. The number of view states in an object movie is defined by the movie file.

SPECIAL CONSIDERATIONS

`QTVRGetViewStateCount` is valid only for object nodes.

QTVRGetViewState

You can use the `QTVRGetViewState` function to get the value of a view state.

```
OSErr QTVRGetViewState (
                    QTVRInstance qtvr,
                    QTVRViewStateType viewStateType,
                    UInt16 *state);
```

`qtvr`	An instance of a QuickTime VR movie.
`viewStateType`	A view state type. See “View State Types” (page 77) for a description of the available view state types.
`state`	On entry, a pointer to a short integer. On exit, that integer is set to the current value of the specified type of view state.
function result	A result code.

DESCRIPTION

The `QTVRGetViewState` function returns, in the `state` parameter, the current value of the view state of the object node specified by the `qtvr` parameter that has the type specified by the `viewStateType` parameter.

SPECIAL CONSIDERATIONS

`QTVRGetViewState` is valid only for object nodes.

SEE ALSO

Use `QTVRSetViewState` (next) to set the value of a view state.

QTVRSetViewState

You can use the `QTVRSetViewState` function to set the value of a view state.

```
OSErr QTVRSetViewState (
                    QTVRInstance qtvr,
                    QTVRViewStateType viewStateType,
                    UInt16 state);
```

`qtvr` An instance of a QuickTime VR movie.

`viewStateType`
A view state type. See "View State Types" (page 77) for a description of the available view state types.

`state` The desired value of the specified type of view state.

function result A result code.

DESCRIPTION

The `QTVRSetViewState` function sets the value of the view state of the object node specified by the `qtvr` parameter that has the type specified by the `viewStateType` parameter to the value specified by the `state` parameter.

SPECIAL CONSIDERATIONS

`QTVRSetViewState` is valid only for object nodes.

SEE ALSO

Use `QTVRGetViewState` (page 135) to get the value of a view state.

QTVRGetAnimationSetting

You can use the `QTVRGetAnimationSetting` function to get the current state of an animation setting for an object node.

```
OSErr QTVRGetAnimationSetting (
                    QTVRInstance qtvr,
                    QTVRObjectAnimationSetting setting,
                    Boolean *enable);
```

`qtvr`	An instance of a QuickTime VR movie.
`setting`	An animation setting. See “Animation Settings” (page 72) for a description of the available animation settings.
`enable`	On entry, a pointer to a Boolean value. On exit, that value is set to `true` if the specified animation setting is currently enabled for the specified object node or to `false` otherwise.
function result	A result code.

DESCRIPTION

The `QTVRGetAnimationSetting` function returns, through the `enable` parameter, the current state of the animation setting specified by the `setting` parameter for the object node specified by the `qtvr` parameter. If `enable` is `true`, the specified setting is currently enabled; otherwise, that setting is disabled.

SPECIAL CONSIDERATIONS

`QTVRGetAnimationSetting` is valid only for object nodes.

SEE ALSO

Use `QTVRSetAnimationSetting` (next) to set the state of an animation setting for an object node.

QTVRSetAnimationSetting

You can use the `QTVRSetAnimationSetting` function to set the state of an animation setting for an object node.

```
OSErr QTVRSetAnimationSetting (
                    QTVRInstance qtvr,
                    QTVRObjectAnimationSetting setting,
                    Boolean enable);
```

`qtvr`	An instance of a QuickTime VR movie.
`setting`	An animation setting. See “Animation Settings” (page 72) for a description of the available animation settings.
`enable`	A Boolean value that indicates whether the specified animation setting is to be enabled for the specified object node (`true`) or disabled (`false`).
function result	A result code.

DESCRIPTION

The `QTVRSetAnimationSetting` function sets the state of the animation setting specified by the `setting` parameter for the object node specified by the `qtvr` parameter to the state specified by the `enable` parameter.

SPECIAL CONSIDERATIONS

`QTVRSetAnimationSetting` is valid only for object nodes.

SEE ALSO

Use `QTVRGetAnimationSetting` (page 137) to get the current state of an animation setting for an object node.

QTVRGetControlSetting

You can use the `QTVRGetControlSetting` function to get the current state of a control setting for an object node.

```
OSErr QTVRGetControlSetting (
                    QTVRInstance qtvr,
                    QTVRControlSetting setting,
                    Boolean *enable);
```

`qtvr`	An instance of a QuickTime VR movie.
`setting`	A control setting. See “Control Settings” (page 74) for a description of the available control settings.
`enable`	On entry, a pointer to a Boolean value. On exit, that value is set to `true` if the specified control setting is currently enabled for the specified object node or to `false` otherwise.
function result	A result code.

DESCRIPTION

The `QTVRGetControlSetting` function returns, through the `enable` parameter, the current state of the control setting specified by the `setting` parameter for the object node specified by the `qtvr` parameter. If `enable` is `true`, the specified setting is currently enabled; otherwise, that setting is disabled.

SPECIAL CONSIDERATIONS

`QTVRGetControlSetting` is valid only for object nodes.

SEE ALSO

Use `QTVRSetControlSetting` (next) to set the state of a control setting for an object node.

QTVRSetControlSetting

You can use the `QTVRSetControlSetting` function to set the state of a control setting for an object node.

```
OSErr QTVRSetControlSetting (
                    QTVRInstance qtvr,
                    QTVRControlSetting setting,
                    Boolean enable);
```

`qtvr`	An instance of a QuickTime VR movie.
`setting`	A control setting. See “Control Settings” (page 74) for a description of the available control settings.
`enable`	A Boolean value that indicates whether the specified control setting is to be enabled for the specified object node (`true`) or disabled (`false`).
function result	A result code.

DESCRIPTION

The `QTVRSetControlSetting` function sets the state of the control setting specified by the `setting` parameter for the object node specified by the `qtvr` parameter to the state specified by the `enable` parameter.

SPECIAL CONSIDERATIONS

`QTVRSetControlSetting` is valid only for object nodes.

SEE ALSO

Use `QTVRGetControlSetting` (page 139) to get the current state of a control setting for an object node.

QTVRGetFrameAnimation

You can use the `QTVRGetFrameAnimation` function to get the current state of frame animation for an object node.

```
Boolean QTVRGetFrameAnimation (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A Boolean value that indicates whether frame animation is enabled for the specified object node (`true`) or not (`false`).

DESCRIPTION

The `QTVRGetFrameAnimation` function returns, as its function result, a Boolean value that indicates the current state of frame animation for the object node specified by the `qtvr` parameter.

SPECIAL CONSIDERATIONS

`QTVRGetFrameAnimation` is valid only for object nodes.

SEE ALSO

Use `QTVREnableFrameAnimation` (next) to set the state of frame animation for an object node.

QTVREnableFrameAnimation

You can use the `QTVREnableFrameAnimation` function to enable or disable frame animation for an object node.

```
OSErr QTVREnableFrameAnimation (QTVRInstance qtvr, Boolean enable);
```

`qtvr`	An instance of a QuickTime VR movie.
`enable`	A Boolean value that indicates whether to enable (`true`) or disable (`false`) frame animation for the specified object node.
function result	A result code.

DESCRIPTION

The `QTVREnableFrameAnimation` function enables or disables the frame animation state for the object node specified by the `qtvr` parameter, according to the value of the `enable` parameter.

The current frame rate (as set by the function `QTVRSetFrameRate`) is unaffected by the state of frame animation of an object node.

SPECIAL CONSIDERATIONS

`QTVREnableFrameAnimation` is valid only for object nodes.

You should use `QTVREnableFrameAnimation` instead of standard QuickTime functions to control object animation.

SEE ALSO

Use `QTVRGetFrameAnimation` (page 141) to get the current state of frame animation for an object node.

QTVRGetViewAnimation

You can use the `QTVRGetViewAnimation` function to get the current state of view animation for an object node.

```
Boolean QTVRGetViewAnimation (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A Boolean value that indicates whether view animation is enabled for the specified object node (`true`) or not (`false`).

DESCRIPTION

The `QTVRGetViewAnimation` function returns, as its function result, a Boolean value that indicates the current state of view animation for the object node specified by the `qtvr` parameter.

SPECIAL CONSIDERATIONS

`QTVRGetViewAnimation` is valid only for object nodes.

SEE ALSO

Use `QTVREnableViewAnimation` (next) to set the state of view animation for an object node.

QTVREnableViewAnimation

You can use the `QTVREnableViewAnimation` function to enable or disable view animation for an object node.

```
OSErr QTVREnableViewAnimation (QTVRInstance qtvr, Boolean enable);
```

`qtvr`	An instance of a QuickTime VR movie.
`enable`	A Boolean value that indicates whether to enable (`true`) or disable (`false`) view animation for the specified object node.
function result	A result code.

DESCRIPTION

The `QTVREnableViewAnimation` function enables or disables the view animation state for the object node specified by the `qtvr` parameter, according to the value of the `enable` parameter.

SPECIAL CONSIDERATIONS

`QTVREnableViewAnimation` is valid only for object nodes.

You should use `QTVREnableViewAnimation` instead of standard QuickTime functions to control object animation.

SEE ALSO

Use `QTVRGetViewAnimation` (page 142) to get the current state of view animation for an object node.

Managing Imaging Characteristics

The QuickTime VR Manager provides functions that you can use to get and set imaging characteristics of a QuickTime VR movie.

QTVRGetVisible

You can use the `QTVRGetVisible` function to get a movie's visibility state.

```
Boolean QTVRGetVisible (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A Boolean value that indicates whether the specified movie is visible (`true`) or not (`false`).

DESCRIPTION

The `QTVRGetVisible` function returns, as its function result, a Boolean value that indicates whether the QuickTime VR movie specified by the `qtvr` parameter is visible (`true`) or not (`false`).

SPECIAL CONSIDERATIONS

`QTVRGetVisible` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRSetVisible` (next) to set the visibility state of a movie.

QTVRSetVisible

You can use the `QTVRSetVisible` function to set a movie's visibility state.

```
OSErr QTVRSetVisible (QTVRInstance qtvr, Boolean visible);
```

`qtvr`	An instance of a QuickTime VR movie.
`visible`	A Boolean value that indicates whether the specified movie is to be visible (`true`) or not (`false`).
function result	A result code.

DESCRIPTION

The `QTVRSetVisible` function sets the visibility state of the QuickTime VR movie specified by the `qtvr` parameter to the state specified by the `visible` parameter. Setting the visibility state to `false` is useful if you want to turn off imaging a QuickTime VR movie without purging the associated data from memory. When a panoramic node's visibility state is `false`, the corrected image is still drawn to the prescreen buffer. You can access the data in that buffer by calling `QTVRSetPrescreenImagingCompleteProc`.

SPECIAL CONSIDERATIONS

`QTVRSetVisible` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRGetVisible` (page 144) to get the visibility state of a movie.

QTVRGetImagingProperty

You can use the `QTVRGetImagingProperty` function to get the current value of an imaging property of a movie.

```
OSErr QTVRGetImagingProperty (
                    QTVRInstance qtvr,
                    QTVRImagingMode imagingMode,
                    UInt32 imagingProperty,
                    SInt32 *propertyValue);
```

`qtvr`	An instance of a QuickTime VR movie.
`imagingMode`	An imaging mode. See "Imaging Modes" (page 63) for a description of the available imaging modes.

`imagingProperty`
: An imaging property. See “Imaging Property Types” (page 64) for a description of the available imaging properties.

`propertyValue`
: On entry, a pointer to a long integer. On exit, that long integer contains the current value of the specified imaging property for the specified mode.

function result
: A result code.

DESCRIPTION

The `QTVRGetImagingProperty` function returns, in the long integer pointed to by the `propertyValue` parameter, the current value of the property specified by the `imagingProperty` parameter when the QuickTime VR movie specified by the `qtvr` parameter is in the mode specified by the `imagingMode` parameter.

SPECIAL CONSIDERATIONS

`QTVRGetImagingProperty` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRSetImagingProperty` (next) to set an imaging property.

QTVRSetImagingProperty

You can use the `QTVRSetImagingProperty` function to set the value of an imaging property of a movie.

```
OSErr QTVRSetImagingProperty (
                    QTVRInstance qtvr,
                    QTVRImagingMode imagingMode,
                    UInt32 imagingProperty,
                    SInt32 propertyValue);
```

`qtvr`
: An instance of a QuickTime VR movie.

`imagingMode`	An imaging mode. See “Imaging Modes” (page 63) for a description of the available imaging modes.
`imagingProperty`	An imaging property. See “Imaging Property Types” (page 64) for a description of the available imaging properties.
`propertyValue`	The desired value for the specified imaging property for the specified mode.
function result	A result code.

DESCRIPTION

The `QTVRSetImagingProperty` function sets the value of the imaging property specified by the `imagingProperty` parameter for the imaging mode specified by the `imagingMode` parameter and the QuickTime VR movie specified by the `qtvr` parameter to the value passed in the `propertyValue` parameter.

Note that default values for all imaging properties can be contained in a QuickTime VR movie file. If no defaults are specified in a movie file, the QuickTime VR Manager uses these values: for static mode, the `kQTVRImagingQuality` property is `codecHighQuality` and `kQTVRImagingDirectDraw` is `true`; for motion mode, the `kQTVRImagingQuality` property is `codecLowQuality` and `kQTVRImagingDirectDraw` is `true`. The default correction mode is `kQTVRFullCorrection` for both static and motion imaging modes.

IMPORTANT

It would look strange to have one correction mode for static imaging and a different correction mode for motion imaging. As a result, you should typically set the `imagingMode` parameter to `kQTVRAllModes` when setting a property of type `kQTVRImagingCorrection`. ▲

SPECIAL CONSIDERATIONS

`QTVRSetImagingProperty` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRGetImagingProperty` (page 145) to get an imaging property.

QTVRUpdate

You can use the `QTVRUpdate` function to force an immediate update of a QuickTime VR movie image.

```
OSErr QTVRUpdate (QTVRInstance qtvr, QTVRImagingMode imagingMode);
```

`qtvr`	An instance of a QuickTime VR movie.
`imagingMode`	An imaging mode. See “Imaging Modes” (page 63) for a description of the available imaging modes. You can specify the `kQTVRCurrentMode` imaging mode to use the current imaging mode.
function result	A result code.

DESCRIPTION

The `QTVRUpdate` function immediately updates the image for the QuickTime VR movie specified by the `qtvr` parameter, without waiting for the next call to `MoviesTask` in your application’s main event loop.

If you plan to call `QTVRUpdate` repeatedly for a movie instance, then for improved performance you should bracket those calls with calls to the `QTVRBeginUpdateStream` and `QTVREndUpdateStream` functions (described next).

SPECIAL CONSIDERATIONS

If you call `QTVRUpdate` after calling `QTVRBeginUpdateStream` but before calling `QTVREndUpdateStream`, the `imagingMode` parameter passed to `QTVRUpdate` must be the same as the `imagingMode` parameter passed to `QTVRBeginUpdateStream`. If you do not specify the same imaging mode to those two functions, an error will result.

SEE ALSO

Listing 2-3 (page 46) illustrates the use of `QTVRUpdate`.

QTVRBeginUpdateStream

You can use the `QTVRBeginUpdateStream` function to begin a stream of immediate updates to a QuickTime VR movie.

```
OSErr QTVRBeginUpdateStream (
                    QTVRInstance qtvr,
                    QTVRImagingMode imagingMode);
```

`qtvr`	An instance of a QuickTime VR movie.
`imagingMode`	An imaging mode. See “Imaging Modes” (page 63) for a description of the available imaging modes.
function result	A result code.

DESCRIPTION

The `QTVRBeginUpdateStream` function configures the QuickTime VR movie specified by the `qtvr` parameter for a stream of immediate updates to its movie image. After calling `QTVRBeginUpdateStream`, you perform the updates by calling `QTVRUpdate`. When you are finished performing the updates, call `QTVREndUpdateStream`. Issuing a stream of image updates in this manner is significantly faster than simply calling `QTVRUpdate` repeatedly (that is, not within a begin/end update sequence).

Each call to `QTVRBeginUpdateStream` must be balanced by a call to `QTVREndUpdateStream`, but you can nest these calls.

SPECIAL CONSIDERATIONS

After you call `QTVRBeginUpdateStream` and before you call `QTVREndUpdateStream`, you must not change any of the QuickTime VR movie’s imaging properties.

Calling `QTVRBeginUpdateStream` locks down large blocks of memory. As a result, you should minimize the amount of time before calling `QTVREndUpdateStream`.

`QTVRBeginUpdateStream` is valid only for panoramic nodes.

SEE ALSO

Use `QTVREndUpdateStream` (next) to end a stream of immediate updates to a QuickTime VR movie.

QTVREndUpdateStream

You can use the `QTVREndUpdateStream` function to end a stream of immediate updates to a QuickTime VR movie.

```
OSErr QTVREndUpdateStream (QTVRInstance qtvr);
```

`qtvr` An instance of a QuickTime VR movie.

function result A result code.

DESCRIPTION

The `QTVREndUpdateStream` function unlocks the memory locked by the matching call to `QTVRBeginUpdateStream` for the QuickTime VR movie specified by the `qtvr` parameter and reverses any other actions performed by that call.

Each call to `QTVRBeginUpdateStream` must be balanced by a call to `QTVREndUpdateStream`, but you can nest these calls. For nested calls, only the final call to `QTVREndUpdateStream` unlocks the memory locked by the first call to `QTVRBeginUpdateStream`.

SPECIAL CONSIDERATIONS

`QTVREndUpdateStream` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRBeginUpdateStream` (page 149) to begin a stream of immediate updates to a QuickTime VR movie.

QTVRSetTransitionProperty

You can use the `QTVRSetTransitionProperty` function to set the value of a transition property.

```
OSErr QTVRSetTransitionProperty (
                    QTVRInstance qtvr,
                    UInt32 transitionType,
                    UInt32 transitionProperty,
                    SInt32 transitionValue);
```

`qtvr` An instance of a QuickTime VR movie.

`transitionType`
A type of transition. See “Transition Type” (page 66) for a description of the available transition type.

`transitionProperty`
A type of transition property. See “Transition Properties” (page 66) for a description of the available transition property types.

`transitionValue`
The desired value for the specified transition property.

function result A result code.

DESCRIPTION

The `QTVRSetTransitionProperty` function sets the value of the transition property whose type is specified by the `transitionType` and `transitionProperty` parameters for the movie specified by the `qtvr` parameter to the value specified by the `transitionValue` parameter.

Note that calling `QTVRSetTransitionProperty` simply sets a transition property’s value; you must still call `QTVREnableTransition` to enable that transition effect.

SPECIAL CONSIDERATIONS

`QTVRSetTransitionProperty` is valid only for panoramic nodes.

SEE ALSO

Use `QTVREnableTransition` (next) to enable a transition effect.

QTVREnableTransition

You can use the `QTVREnableTransition` function to enable or disable a transition effect.

```
OSErr QTVREnableTransition (
                    QTVRInstance qtvr,
                    UInt32 transitionType,
                    Boolean enable);
```

`qtvr` An instance of a QuickTime VR movie.

`transitionType`
A type of transition property. See “Transition Type” (page 66) for a description of the available transition type.

`enable` A Boolean value that indicates whether the specified transition property is to be enabled (`true`) or disabled (`false`).

function result A result code.

DESCRIPTION

The `QTVREnableTransition` function enables or disables the transition property specified by the `transitionType` parameter for the movie specified by the `qtvr` parameter, as indicated by the value of the `enable` parameter. Once a transition effect is enabled, it is used at the appropriate time until it is disabled by a subsequent call to `QTVREnableTransition`.

SPECIAL CONSIDERATIONS

`QTVREnableTransition` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRSetTransitionProperty` (page 151) to set the value of a transition property.

Converting Angles and Points

The QuickTime VR Manager provides routines that you can use to convert mathematical entities and perform other utility operations.

QTVRGetAngularUnits

You can use the `QTVRGetAngularUnits` function to get the type of unit currently used when specifying angles.

```
QTVRAngularUnits QTVRGetAngularUnits (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A constant that indicates the type of angular units currently in use. See “Angular Unit Types” (page 59) for a description of the available angular unit types.

DESCRIPTION

The `QTVRGetAngularUnits` function returns, as its function result, a constant that indicates the type of angular unit currently used by the movie instance specified by the `qtvr` parameter. Angular values you pass to other QuickTime VR functions (for example, `QTVRSetPanAngle`) are interpreted in those units.

SEE ALSO

Use `QTVRSetAngularUnits` (next) to set the type of angular unit used by a QuickTime VR movie.

QTVRSetAngularUnits

You can use the `QTVRSetAngularUnits` function to set the type of unit used when specifying angles.

```
OSErr QTVRSetAngularUnits (QTVRInstance qtvr, QTVRAngularUnits units);
```

`qtvr`	An instance of a QuickTime VR movie.
`units`	A constant that indicates the type of angular units to use. See "Angular Unit Types" (page 59) for a description of the available angular unit types.
function result	A result code.

DESCRIPTION

The `QTVRSetAngularUnits` function sets the type of angular units to be used in all subsequent QuickTime VR Manager calls for the QuickTime VR movie specified by the `qtvr` parameter to the unit type specified by the `units` parameter.

SEE ALSO

Use `QTVRGetAngularUnits` (page 153) to get the type of angular unit used by a QuickTime VR movie. Listing 2-2 (page 44) illustrates the use of `QTVRSetAngularUnits`.

QTVRPtToAngles

You can use the `QTVRPtToAngles` function to get the pan and tilt angles of a point.

```
OSErr QTVRPtToAngles (
                    QTVRInstance qtvr,
                    Point pt,
                    float *panAngle,
                    float *tiltAngle);
```

`qtvr`	An instance of a QuickTime VR movie.

`pt`	A point, in the local coordinates of the graphics world of the specified movie.
`panAngle`	On entry, a pointer to a floating-point value. On exit, that value contains the pan angle of the specified point.
`tiltAngle`	On entry, a pointer to a floating-point value. On exit, that value contains the tilt angle of the specified point.
function result	A result code.

DESCRIPTION

For a panorama, each point in the current view corresponds to a particular pan and tilt angle, with the point at the center of the view corresponding to the panorama's current pan and tilt angle. The `QTVRPtToAngles` function returns, in the floating-point values pointed to by the `panAngle` and `tiltAngle` parameters, the pan and tilt angles of the point specified by the `pt` parameter.

SPECIAL CONSIDERATIONS

`QTVRPtToAngles` is valid only for panoramic nodes.

QTVRCoordToAngles

You can use the `QTVRCoordToAngles` function to get the pan and tilt angles of a floating-point coordinate in a panorama.

```
OSErr QTVRCoordToAngles (
                    QTVRInstance qtvr,
                    QTVRFloatPoint *coord,
                    float *panAngle,
                    float *tiltAngle);
```

`qtvr`	An instance of a QuickTime VR movie.
`coord`	On entry, a pointer to a `QTVRFloatPoint` structure that specifies a coordinate in the full panorama.
`panAngle`	On entry, a pointer to a floating-point value. On exit, that value contains the pan angle of the specified coordinate.

`tiltAngle`	On entry, a pointer to a floating-point value. On exit, that value contains the tilt angle of the specified coordinate.
function result	A result code.

DESCRIPTION

The `QTVRCoordToAngles` function returns, in the floating-point values pointed to by the `panAngle` and `tiltAngle` parameters, the pan and tilt angles of the point specified by the `coord` parameter. This function is useful for setting up angles in a back buffer imaging procedure; if you know a coordinate in the back buffer, you can call `QTVRCoordToAngles` to get the corresponding angles.

SPECIAL CONSIDERATIONS

`QTVRCoordToAngles` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRAnglesToCoord` (next) to get a floating-point coordinate from a pair of pan and tilt angles.

QTVRAnglesToCoord

You can use the `QTVRAnglesToCoord` function to get a floating-point coordinate determined by a pair of pan and tilt angles.

```
OSErr QTVRAnglesToCoord (
                    QTVRInstance qtvr,
                    float panAngle,
                    float tiltAngle,
                    QTVRFloatPoint *coord);
```

`qtvr`	An instance of a QuickTime VR movie.
`panAngle`	A pan angle.
`tiltAngle`	A tilt angle.

`coord`	On entry, a pointer to a `QTVRFloatPoint` structure. On exit, that structure is set to the coordinate of the specified movie that corresponds to the specified pan and tilt angles.
function result	A result code.

DESCRIPTION

The `QTVRAnglesToCoord` function returns, in the `QTVRFloatPoint` structure pointed to by the `coord` parameter, the coordinates of the point in the full panorama of the movie specified by the `qtvr` parameter that corresponds to the pan and tilt angles specified by the `panAngle` and `tiltAngle` parameters.

SPECIAL CONSIDERATIONS

`QTVRAnglesToCoord` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRCoordToAngles` (page 155) to get a pair of pan and tilt angles from a floating-point coordinate.

QTVRPanToColumn

You can use the `QTVRPanToColumn` function to get the column number in the object image array that corresponds to a pan angle.

```
short QTVRPanToColumn (QTVRInstance qtvr, float panAngle);
```

`qtvr`	An instance of a QuickTime VR movie.
`panAngle`	A pan angle.
function result	The column number in the object image array that corresponds to the specified pan angle.

DESCRIPTION

The `QTVRPanToColumn` function returns, as its function result, the zero-based column number in the current object image array that corresponds to the pan angle specified by the `panAngle` parameter.

SPECIAL CONSIDERATIONS

`QTVRPanToColumn` is valid only for object nodes.

QTVRColumnToPan

You can use the `QTVRColumnToPan` function to get the pan angle that corresponds to a column number in the object image array.

```
float QTVRColumnToPan (QTVRInstance qtvr, short column);
```

`qtvr`	An instance of a QuickTime VR movie.
`column`	A column number.
function result	The pan angle that corresponds to the specified column number in the object image array.

DESCRIPTION

The `QTVRColumnToPan` function returns, as its function result, the pan angle that corresponds to the zero-based column number in the object image array specified by the `column` parameter.

SPECIAL CONSIDERATIONS

`QTVRColumnToPan` is valid only for object nodes.

QTVRTiltToRow

You can use the `QTVRTiltToRow` function to get the row number in the object image array that corresponds to a tilt angle.

```
short QTVRTiltToRow (QTVRInstance qtvr, float tiltAngle);
```

`qtvr`	An instance of a QuickTime VR movie.
`tiltAngle`	A tilt angle.
function result	The row number in the object image array that corresponds to the specified tilt angle.

DESCRIPTION

The `QTVRTiltToRow` function returns, as its function result, the zero-based row number in the current object image array that corresponds to the tilt angle specified by the `tiltAngle` parameter.

SPECIAL CONSIDERATIONS

`QTVRTiltToRow` is valid only for object nodes.

QTVRRowToTilt

You can use the `QTVRRowToTilt` function to get the tilt angle that corresponds to a row number in the object image array.

```
float QTVRRowToTilt (QTVRInstance qtvr, short row);
```

`qtvr`	An instance of a QuickTime VR movie.
`row`	A row number.
function result	The tilt angle that corresponds to the specified row number in the object image array.

DESCRIPTION

The `QTVRRowToTilt` function returns, as its function result, the tilt angle that corresponds to the zero-based row number in the object image array specified by the `row` parameter.

SPECIAL CONSIDERATIONS

`QTVRRowToTilt` is valid only for object nodes.

QTVRWrapAndConstrain

You can use the `QTVRWrapAndConstrain` function to preflight a change in the viewing or control characteristics of an object or panoramic node.

```
OSErr QTVRWrapAndConstrain (
                    QTVRInstance qtvr,
                    short kind,
                    float value,
                    float *result);
```

`qtvr`	An instance of a QuickTime VR movie.
`kind`	A constraint type. See “Constraint Types” (page 61) for a description of the available constraint types.
`value`	The desired value of the specified viewing characteristic.
`result`	On exit, the value to which the specified viewing characteristic would be set if it were changed.
function result	A result code.

DESCRIPTION

The `QTVRWrapAndConstrain` function returns, in the `result` parameter, the constrained or wrapped value that would result from setting the viewing or control characteristic specified by the `kind` parameter to the value specified by the `value` parameter. For example, if the `kind` parameter is set to `kQTVRPan`, then `QTVRWrapAndConstrain` returns the value that would result from calling the

QTVRSetPanAngle function with its panAngle parameter set to value. Similarly, you can use QTVRWrapAndConstrain to find the current bounds of the view center. QTVRWrapAndConstrain takes into account the current constraints and wrapping modes of the node specified by the qtvr parameter.

QTVRWrapAndConstrain does not change the current view or other settings of the specified object or panorama.

Managing QuickTime VR Movie Interaction

The QuickTime VR Manager provides functions that you can use to augment the normal user interaction handling provided by a QuickTime movie controller. For example, you can install an action filter function to handle movie controller actions; see *QuickTime 3 Reference* for details.

QTVRSetEnteringNodeProc

You can use the QTVRSetEnteringNodeProc function to install or remove a node-entering procedure.

```
OSErr QTVRSetEnteringNodeProc (
                    QTVRInstance qtvr,
                    QTVREnteringNodeUPP enteringNodeProc,
                    SInt32 refCon,
                    UInt32 flags);
```

qtvr — An instance of a QuickTime VR movie.

enteringNodeProc — A universal procedure pointer for a node-entering procedure. See “MyEnteringNodeProc” (page 183) for information about node-entering procedures.

refCon — A reference constant. This value is passed to the specified node-entering procedure.

flags — Unused. Set this parameter to 0.

function result — A result code.

DESCRIPTION

The `QTVRSetEnteringNodeProc` function installs the procedure specified by the `enteringNodeProc` parameter as a node-entering procedure for the QuickTime VR movie specified by the `qtvr` parameter. Your procedure is called whenever a node is entered (either in response to user actions or in response to QuickTime VR Manager functions that change nodes). The reference constant specified by the `refCon` parameter is passed unchanged to that node-entering procedure.

To remove a previously installed node-entering procedure, set `enteringNodeProc` to `nil`.

SEE ALSO

Use `QTVRSetLeavingNodeProc` (next) to install or remove a node-leaving procedure.

QTVRSetLeavingNodeProc

You can use the `QTVRSetLeavingNodeProc` function to install or remove a node-leaving procedure.

```
OSErr QTVRSetLeavingNodeProc (
                    QTVRInstance qtvr,
                    QTVRLeavingNodeUPP leavingNodeProc,
                    SInt32 refCon,
                    UInt32 flags);
```

`qtvr` An instance of a QuickTime VR movie.

`leavingNodeProc` A universal procedure pointer for a node-leaving procedure. See “MyLeavingNodeProc” (page 184) for information about node-leaving procedures.

`refCon` A reference constant. This value is passed to the specified node-leaving procedure.

`flags` Unused. Set this parameter to 0.

function result A result code.

DESCRIPTION

The `QTVRSetLeavingNodeProc` function installs the procedure specified by the `leavingNodeProc` parameter as a node-leaving procedure for the QuickTime VR movie specified by the `qtvr` parameter. Your procedure is called whenever a node is left (either in response to user actions or in response to QuickTime VR Manager functions that change nodes). The reference constant specified by the `refCon` parameter is passed unchanged to that node-leaving procedure.

To remove a previously installed node-leaving procedure, set `leavingNodeProc` to `nil`.

SEE ALSO

Use `QTVRSetEnteringNodeProc` (page 161) to install or remove a node-entering procedure.

QTVRGetInteractionProperty

You can use the `QTVRGetInteractionProperty` function to get the value of an interaction property.

```
OSErr QTVRGetInteractionProperty (
                    QTVRInstance qtvr,
                    UInt32 property,
                    void *value);
```

`qtvr`	An instance of a QuickTime VR movie.
`property`	An interaction property type. See “Interaction Property Types” (page 68) for a description of the available types of interaction properties.
`value`	On entry, a pointer to a block of memory. On exit, that memory contains the current value of the specified interaction property.
function result	A result code.

DESCRIPTION

The `QTVRGetInteractionProperty` function returns, in the block of memory pointed to by the `value` parameter, the current value of the property specified by the `property` parameter for the QuickTime VR movie specified by the `qtvr` parameter. That block of memory must be large enough to hold the returned value.

SEE ALSO

Use `QTVRSetInteractionProperty` (next) to set an interaction property.

QTVRSetInteractionProperty

You can use the `QTVRSetInteractionProperty` function to set the value of an interaction property.

```
OSErr QTVRSetInteractionProperty (
                    QTVRInstance qtvr,
                    UInt32 property,
                    void *value);
```

`qtvr`	An instance of a QuickTime VR movie.
`property`	An interaction property type. See “Interaction Property Types” (page 68) for a description of the available types of interaction properties.
`value`	The desired value of the specified interaction property.
function result	A result code.

DESCRIPTION

The `QTVRSetInteractionProperty` function sets the value of the interaction property of the type specified by the `property` parameter for the movie specified by the `qtvr` parameter to the value specified by the `value` parameter. For types that occupy 32 or fewer bits of memory, you pass the desired value itself (cast to a `void *`) in the `value` parameter. For structures and floating-point values, you must pass a pointer to the desired value in the `value` parameter.

Note
Floating-point values are usually stored as 32-bit values, but compilers differ in how they pass floating-point values as parameters; as a result, `QTVRSetInteractionProperty` demands that floating-point values always be passed by reference. ◆

SEE ALSO

Use `QTVRGetInteractionProperty` (page 163) to get an interaction property.

QTVRReplaceCursor

You can use the `QTVRReplaceCursor` function to replace any of the standard QuickTime VR cursors with your own custom cursors.

```
OSErr QTVRReplaceCursor (QTVRInstance qtvr, CursorRecord *cursRecord);
```

`qtvr`	An instance of a QuickTime VR movie.
`cursRecord`	A pointer to a cursor record. See "Cursor Record" (page 87) for a description of the cursor record.
function result	A result code.

DESCRIPTION

The `QTVRReplaceCursor` function replaces one or more of the standard QuickTime VR cursors associated with the instance specified by the `qtvr` parameter with the cursors specified in the cursor record pointed to by the `cursRecord` parameter. If the `type` field of the specified cursor record is `kQTVRUseDefaultCursor`, the default cursor for the given resource ID is reloaded; in this case, the `handle` field of that record should be set to `nil`.

`QTVRReplaceCursor` replaces the standard cursors only for the specified QuickTime VR movie instance. To replace the standard cursors for all QuickTime VR movie instances you create, you need to call `QTVRReplaceCursor` for each such instance.

Note
QuickTime VR 2.1 makes a copy of the cursor handle specified in the cursor record. The application is responsible for disposing of its own cursor handle.

Determining Viewing Limits and Constraints

The QuickTime VR Manager provides functions that you can use to get the physical viewing limits of a movie and to get and set a movie's constraints.

QTVRGetViewingLimits

You can use the `QTVRGetViewingLimits` function to get the current viewing limits of a QuickTime VR movie.

```
OSErr QTVRGetViewingLimits (
                    QTVRInstance qtvr,
                    UInt16 kind,
                    float *minValue,
                    float *maxValue);
```

`qtvr`	An instance of a QuickTime VR movie.
`kind`	The type of viewing limits to be returned. See "Constraint Types" (page 61) for a description of the available types of viewing limits.
`minValue`	On entry, a pointer to a floating-point value. On exit, the minimum viewing limit of the specified type is copied into that value.
`maxValue`	On entry, a pointer to a floating-point value. On exit, the maximum viewing limit of the specified type is copied into that value.
function result	A result code.

DESCRIPTION

The `QTVRGetViewingLimits` function returns, in the floating-point values pointed to by the `minValue` and `maxValue` parameters, the current minimum and maximum values for angles whose type is specified by the `kind` parameter.

The maximum field of view of a panoramic node can be limited by the size of the back buffer and the current aspect ratio of the movie's graphics world.

The values returned by `QTVRGetViewingLimits` are unaffected by the current control settings.

`QTVRGetViewingLimits` returns information about the physical viewing limits of a panorama or object. To get information about the current viewing constraints, use `QTVRGetConstraints` (page 168).

QTVRGetConstraintStatus

You can use the `QTVRGetConstraintStatus` function to get the set of constraints active for the current view.

```
UInt32 QTVRGetConstraintStatus (QTVRInstance qtvr);
```

`qtvr`	An instance of a QuickTime VR movie.
function result	A long integer whose bits are set or cleared to represent the currently active constraints of the specified QuickTime VR movie.

DESCRIPTION

The `QTVRGetConstraintStatus` function returns, as its function result, a long integer whose bits encode the constraints currently active for the QuickTime VR movie specified by the `qtvr` parameter. See "Viewing Constraints" (page 70) for a description of the available constraints.

The values returned by `QTVRGetConstraintStatus` are unaffected by the current control settings.

QTVRGetConstraints

You can use the `QTVRGetConstraints` function to get the current constraints of a movie.

```
OSErr QTVRGetConstraints (
                    QTVRInstance qtvr,
                    UInt16 kind,
                    float *minValue,
                    float *maxValue);
```

`qtvr`	An instance of a QuickTime VR movie.
`kind`	The type of constraints to be returned. See “Constraint Types” (page 61) for a description of the available types of constraints.
`minValue`	On entry, a pointer to a floating-point value. On exit, the current minimum constraint of the specified type is copied into that value.
`maxValue`	On entry, a pointer to a floating-point value. On exit, the current maximum constraint of the specified type is copied into that value.
function result	A result code.

DESCRIPTION

The `QTVRGetConstraints` function returns, in the floating-point values pointed to by the `minValue` and `maxValue` parameters, the current minimum and maximum constraints of the type specified by the `kind` parameter.

The values returned by `QTVRGetConstraints` are unaffected by the current control settings.

SEE ALSO

Use `QTVRSetConstraints` (next) to set a movie’s minimum and maximum constraints.

QTVRSetConstraints

You can use the `QTVRSetConstraints` function to set the constraints of a movie.

```
OSErr QTVRSetConstraints (
                    QTVRInstance qtvr,
                    UInt16 kind,
                    float minValue,
                    float maxValue);
```

`qtvr`	An instance of a QuickTime VR movie.
`kind`	The type of constraint to set. See “Constraint Types” (page 61) for a description of the available types of constraints.
`minValue`	A floating-point value that contains the desired minimum constraint of the specified type.
`maxValue`	A floating-point value that contains the desired maximum constraint of the specified type.
function result	A result code.

DESCRIPTION

The `QTVRSetConstraints` function sets the minimum and maximum constraints of the type specified by the `kind` parameter to the values specified by the `minValue` and `maxValue` parameters. Note that when you want to specify a pan angle constraint, the `minValue` and `maxValue` parameters should be specified so that a clockwise sweep from `minValue` to `maxValue` selects the desired angular expanse. For example, to constrain panning in the 90-degree expanse that spreads out 45 degrees on each side of the pan angle 0 degrees, you should set the `minValue` parameter to 315 degrees and the `maxValue` parameter to 45 degrees. Similarly, to constrain panning in the remaining 270-degree expanse, you should set the `minValue` parameter to 45 degrees and the `maxValue` parameter to 315 degrees.

The values passed to `QTVRSetConstraints` are unaffected by the current control settings.

SEE ALSO

Use `QTVRGetConstraints` (page 168) to get a movie’s minimum and maximum constraints.

Managing Memory

The QuickTime VR Manager provides functions that you can use to help manage the memory used by QuickTime VR.

QTVRGetAvailableResolutions

You can use the `QTVRGetAvailableResolutions` function to get the image resolutions present in the current node.

```
OSErr QTVRGetAvailableResolutions (
                    QTVRInstance qtvr,
                    UInt16 *resolutionsMask);
```

`qtvr` An instance of a QuickTime VR movie.

`resolutionsMask`
On entry, a pointer to an unsigned short integer. On exit, that integer is set to a bitmask that encodes the image resolutions available at the current node.

function result A result code.

DESCRIPTION

The `QTVRGetAvailableResolutions` function returns, in the unsigned short integer pointed to by the `resolutionsMask` parameter, a bitmask that encodes the image resolutions available at the current node of the QuickTime VR movie specified by the `qtvr` parameter.

A single node can contain multiple resolutions of a panorama or an object. The lowest order bit is always set and corresponds to the base resolution of the node. Each succeeding bit corresponds to a resolution that is half that (both horizontally and vertically) of the preceding bit. If an image with a resolution is present in the current node, the corresponding bit is set.

QTVRGetBackBufferMemInfo

QuickTime VR maintains an internal back buffer for caching panoramic images. You can use the `QTVRGetBackBufferMemInfo` function to get information about the size of the back buffer that would be required for caching a panoramic image of a specified pixel format, geometry, and resolution. See “Pixel Formats” (page 82) for more information.

```
OSErr QTVRGetBackBufferMemInfo (
                    QTVRInstance qtvr,
                    UInt16 geometry,
                    UInt16 resolution,
                    UInt32 cachePixelFormat,
                    SInt32 *minCacheBytes,
                    SInt32 *suggestedCacheBytes,
                    SInt32 *fullCacheBytes);
```

`qtvr` An instance of a QuickTime VR movie.

`geometry` The geometry selector specifies the type and orientation of the panorama data. Only the vertical cylinder geometry is used in the current version of QuickTime VR; see “Geometry Selectors” (page 84).

`resolution` The resolution for which the information is desired; see “Resolutions” (page 83).

`cachePixelFormat`
The desired pixel format for the back buffer. This value should be one of the defined pixel formats. See “Pixel Formats” (page 82) for more information.

`minCacheBytes` On entry, a pointer to a long integer. On exit, that long integer is set to the minimum size, in bytes, of the back buffer required to display the specified panorama with a severely limited maximum field of view. Set this parameter to `nil` to prevent this information from being returned.

`suggestedCacheBytes`
On entry, a pointer to a long integer. On exit, that long integer is set to the minimum size, in bytes, of the back buffer required to

display the specified panorama with full wide-angle zooming. Set this parameter to `nil` to prevent this information from being returned.

`fullCacheBytes`

On entry, a pointer to a long integer. On exit, that long integer is set to the minimum size, in bytes, of the back buffer required to have the entire panorama in memory at once. That is the default size of the panorama back buffer. Set this parameter to `nil` to prevent this information from being returned.

function result A result code.

DESCRIPTION

The `QTVRGetBackBufferMemInfo` function returns information about the size of the back buffer that would be required to hold some or all of the panoramic image associated with the movie specified by the `qtvr` parameter. This is a "what-if" function: you specify a resolution and a pixel format, and `QTVRGetBackBufferMemInfo` returns several buffer sizes. You can use this information, in conjunction with the `QTVRSetBackBufferPrefs` function, to exercise some control over the size of the back buffer.

The resolution at which an image is to be displayed is specified by the `resolution` parameter. You can use a resolution that is not in the movie file. Relative to that resolution and the pixel depth determined by the `cachePixelFormat` parameter, the `QTVRGetBackBufferMemInfo` function returns, through the `minCacheBytes` parameter, the minimum size of the buffer needed to display the movie. Using a buffer of that size, however, may result in a severely limited maximum field of view. You can call the `QTVRGetViewingLimits` function to determine the actual maximum field of view.

To allow full wide-angle zooming, you should use a buffer whose size is specified by either the `suggestedCacheBytes` parameter or the `fullCacheBytes` parameter.

SPECIAL CONSIDERATIONS

`QTVRGetBackBufferMemInfo` is valid only for panoramic nodes.

QTVRGetBackBufferSettings

You can use the `QTVRGetBackBufferSettings` function to get information about the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

```
OSErr QTVRGetBackBufferSettings (
                    QTVRInstance qtvr,
                    UInt16 *geometry,
                    UInt16 *resolution,
                    UInt32 *cachePixelFormat,
                    SInt16 *cacheSize);
```

`qtvr` An instance of a QuickTime VR movie.

`geometry` The geometry selector specifies the type and orientation of the panorama data. Only the vertical cylinder geometry is used in the current version of QuickTime VR; see “Geometry Selectors” (page 84).

`resolution` On entry, a pointer to an unsigned short integer. On exit, that integer is set to the index of the current image resolution. See “Resolutions” (page 83) for a description of the available resolutions.

`cachePixelFormat` On entry, a pointer to a long integer. On exit, that long integer is set to the pixel format of the current panorama back buffer. See “Pixel Formats” (page 82) for more information.

`cacheSize` On entry, a pointer to a short integer. On exit, that integer is set to a value that describes the size of the current panorama back buffer. See “Cache Sizes” (page 84).

function result A result code.

DESCRIPTION

The `QTVRGetBackBufferSettings` function returns, through the `resolution` parameter, the index of the current resolution for the QuickTime VR movie specified by the `qtvr` parameter. The index indicates which bit in the mask value returned by `QTVRGetAvailableResolutions` specifies the current resolution. For example, if the returned index is 1, the base resolution is being used. If the returned index is 2, then a resolution of half the base resolution is being used.

QTVRGetBackBufferSettings also returns the pixel format and the cache size in the cachePixelFormat and cacheSize parameters, respectively.

The QuickTime VR file might not contain an image track corresponding to the resolution indicated by the resolution value returned. The QuickTime VR Manager may have set a lower resolution because memory is low, or the resolution may have been set by a call to the QTVRSetBackBufferPrefs function.

SPECIAL CONSIDERATIONS

QTVRGetBackBufferSettings is valid only for panoramic nodes.

SEE ALSO

Use QTVRSetBackBufferPrefs (next) to set the resolution, cache depth, and cache size of the panorama back buffer maintained internally by QuickTime VR for caching an image. Use QTVRGetAvailableResolutions (page 170) to determine which resolutions are supported by a node. Use QTVRGetBackBufferMemInfo (page 171) to determine the memory requirements for the preferred settings.

QTVRSetBackBufferPrefs

You can use the QTVRSetBackBufferPrefs function to set the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

```
OSErr QTVRSetBackBufferPrefs (
                    QTVRInstance qtvr,
                    UInt16 geometry,
                    UInt16 resolution,
                    UInt32 cachePixelFormat,
                    SInt16 cacheSize);
```

qtvr — An instance of a QuickTime VR movie.

geometry — The geometry selector specifies the type and orientation of the panorama data. Only the vertical cylinder geometry is used in the current version of QuickTime VR; see “Geometry Selectors” (page 84).

`resolution` The desired image resolution; see "Resolutions" (page 83).

`cachePixelFormat` The desired pixel format for the back buffer. This value should be one of the defined pixel formats. See "Pixel Formats" (page 82) for more information.

`cacheSize` The desired size for the panorama back buffer. See "Cache Sizes" (page 84) for constants you can use to specify a cache size.

function result A result code.

DESCRIPTION

The `QTVRSetBackBufferPrefs` function sets the resolution, pixel format, and size of the panorama back buffer for the movie specified by the `qtvr` parameter to the values specified by the `resolution`, `cachePixelFormat`, and `cacheSize` parameters. You can specify a resolution that isn't contained in the movie file; if you do so, QuickTime VR takes the highest resolution image in the file and reduces it to fit into the specified buffer size.

If you specify an unsupported pixel format, the `QTVRSetBackBufferPrefs` function may return an error. See "Pixel Formats" (page 82) for more information.

SPECIAL CONSIDERATIONS

`QTVRSetBackBufferPrefs` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRGetAvailableResolutions` (page 170) to determine which resolutions are supported by a node. Use `QTVRGetBackBufferMemInfo` (page 171) to determine the memory requirements for the preferred settings.

Accessing Image Buffers

The QuickTime VR Manager provides functions that you can use to access the two internal buffers used by QuickTime VR when it's imaging a panorama.

QTVRSetPrescreenImagingCompleteProc

You can use the `QTVRSetPrescreenImagingCompleteProc` function to install or remove a prescreen buffer imaging completion procedure.

```
OSErr QTVRSetPrescreenImagingCompleteProc (
                    QTVRInstance qtvr,
                    ImagingCompleteUPP imagingCompleteProc,
                    SInt32 refCon,
                    UInt32 flags);
```

`qtvr` An instance of a QuickTime VR movie.

`imagingCompleteProc`
A universal procedure pointer for a prescreen buffer imaging completion procedure. See “MyImagingCompleteProc” (page 185) for information about prescreen buffer imaging completion procedures.

`refCon` A reference constant. This value is passed to the specified prescreen buffer imaging completion procedure.

`flags` You can use `kQTVRPreScreenEveryIdle` to cause a draw attempt on every idle passed to the movie controller. See “Flags Value for Imaging Completion Procedure” (page 59).

function result A result code.

DESCRIPTION

The `QTVRSetPrescreenImagingCompleteProc` function installs the procedure specified by the `imagingCompleteProc` parameter as a prescreen buffer imaging completion procedure for the QuickTime VR movie specified by the `qtvr` parameter. Your procedure is called whenever QuickTime VR finishes drawing an image into the prescreen buffer. The reference constant specified by the `refCon` parameter is passed unchanged to that prescreen buffer imaging completion procedure.

To remove a previously installed prescreen buffer imaging completion procedure, set `imagingCompleteProc` to `nil`.

SPECIAL CONSIDERATIONS

`QTVRSetPrescreenImagingCompleteProc` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRSetBackBufferImagingProc` (next) to install or remove a back buffer imaging procedure.

QTVRSetBackBufferImagingProc

You can use the `QTVRSetBackBufferImagingProc` function to install or remove a back buffer imaging procedure.

```
OSErr QTVRSetBackBufferImagingProc (
                    QTVRInstance qtvr,
                    BackBufferImagingUPP backBufferImagingProc,
                    UInt16 numAreas,
                    AreaOfInterest *areasOfInterest,
                    SInt32 refCon);
```

`qtvr` An instance of a QuickTime VR movie.

`backBufferImagingProc`
A universal procedure pointer for a back buffer imaging procedure. See “MyBackBufferImagingProc” (page 186) for information about back buffer imaging procedures.

`numAreas` The number of area of interest structures in the array pointed to by the `areasOfInterest` parameter.

`areasOfInterest`
A pointer to an array of area of interest structures. See “Area of Interest Structure” (page 87).

`refCon` A reference constant. This value is passed to the specified back buffer imaging procedure.

function result A result code.

DESCRIPTION

The `QTVRSetBackBufferImagingProc` function installs the procedure specified by the `backBufferImagingProc` parameter as a back buffer imaging procedure for the panoramic node specified by the `qtvr` parameter. You can use that procedure to draw directly into the back buffer.

The `areasOfInterest` parameter is a pointer to an array of area of interest structures that define the rectangular areas about which you want your back buffer imaging procedure to be notified. Your procedure is called for each area of interest as it becomes visible or not visible. You indicate when you want your procedure to be called for a particular area of interest by setting flags in the `flags` field in the corresponding area of interest structure.

IMPORTANT

The QuickTime VR Manager version 2.1 supports only one area of interest in this array. Future versions will support multiple areas of interest. ▲

Note that coordinates in the back buffer are dependent on the current correction mode; as a result, you need to indicate the area you're interested in drawing into by specifying a pan angle and tilt angle to determine the upper-left corner of the area and a height and width relative to that corner. (Specifying a height and width instead of a second pair of pan and tilt angles for the bottom-right coordinate allows the rectangle to wrap around the edge of the panorama.)

The width of the area of interest is limited by the size of the back buffer. If the back buffer is less than the full cache size, then the area of interest can be no wider than half the size of the back buffer. (For vertical cylinder geometries, limiting factor would be the height of the buffer.) For a full cache back buffer, the width of the area of interest can be the full size of the buffer. If the width limit is exceeded, `QTVRSetBackBufferImagingProc` will return `constraintReachedErr`.

To remove a previously installed back buffer imaging procedure, set `backBufferImagingProc` to `nil`.

SPECIAL CONSIDERATIONS

`QTVRSetBackBufferImagingProc` is valid only for panoramic nodes.

SEE ALSO

Use `QTVRSetPrescreenImagingCompleteProc` (page 176) to install or remove a prescreen buffer imaging completion procedure.

QTVRRefreshBackBuffer

You can use the `QTVRRefreshBackBuffer` function to refresh the back buffer.

```
OSErr QTVRRefreshBackBuffer (QTVRInstance qtvr, UInt32 flags);
```

`qtvr`	An instance of a QuickTime VR movie.
`flags`	Unused. Set this parameter to 0.
function result	A result code.

DESCRIPTION

The `QTVRRefreshBackBuffer` function refreshes some or all of the back buffer associated with the QuickTime VR movie specified by the `qtvr` parameter by reloading the appropriate data from the diced frames in the panorama image track.

You can call `QTVRRefreshBackBuffer` either in a back buffer imaging procedure or elsewhere in your application. If you call `QTVRRefreshBackBuffer` in a back buffer imaging procedure, only the current rectangle (that is, the rectangle specified by the procedure's `drawRect` parameter) is refreshed. If you call `QTVRRefreshBackBuffer` outside of a back buffer imaging procedure, all areas of interest specified in the most recent call to `QTVRSetBackBufferImagingProc` are refreshed.

SPECIAL CONSIDERATIONS

`QTVRRefreshBackBuffer` is valid only for panoramic nodes.

SEE ALSO

See "MyBackBufferImagingProc" (page 186) for information about back buffer imaging procedures.

Application-Defined Routines

This section describes the routines your application or other software component might need to define when using the QuickTime VR Manager.

Mouse Over Hot Spot Procedure

Your application can define a mouse over hot spot procedure that is called when the cursor is over a hot spot.

MyMouseOverHotSpotProc

The `mouseOverHotSpotProc` parameter to the `QTVRSetMouseOverHotSpotProc` function specifies an application-defined mouse over hot spot procedure.

```
pascal OSErr MyMouseOverHotSpotProc (
                    QTVRInstance qtvr,
                    UInt32 hotSpotID,
                    UInt32 flags,
                    SInt32 refCon);
```

`qtvr`	An instance of a QuickTime VR movie.
`hotSpotID`	The ID of the hot spot over which the cursor has been moved.
`flags`	A hot spot action selector. See “Hot Spot Action Selectors” (page 59) for a description of the available selectors.
`refCon`	A reference constant. This is the same value that your application passed in the `refCon` parameter when it called `QTVRSetMouseOverHotSpotProc` to install this callback routine.
function result	A result code that indicates whether QuickTime VR should perform the actions it normally performs when the cursor is over a hot spot (`noErr`) or not (any nonzero value).

DESCRIPTION

Your `MyMouseOverHotSpotProc` routine is called whenever the user moves the cursor over a hot spot, keeps it over a hot spot, or moves it out of a hot spot. The user action that caused your routine to be called is specified by the hot spot action selector passed in the `flags` parameter. For instance, when the cursor is first moved over a hot spot, your routine is called with `flags` set to `kQTVRHotSpotEnter`. Similarly, until the cursor is moved back off that hot spot, your routine is called repeatedly with `flags` set to `kQTVRHotSpotWithin`. Your routine should perform whatever actions it likes and return the value `noErr` if you want QuickTime VR to perform whatever actions it normally performs in that circumstance. Your routine should return any nonzero value to suppress those actions.

IMPORTANT

Your mouse over hot spot procedure is called only for enabled hot spots. ▲

SEE ALSO

Use `QTVRSetMouseOverHotSpotProc` (page 109) to install or remove a mouse over hot spot procedure. Use `QTVREnableHotSpot` (page 108) to enable or disable a hot spot.

QuickTime VR Intercept Procedure

You can install an intercept procedure that is executed immediately before the QuickTime VR Manager function it is intercepting.

MyInterceptProc

You can define a routine to intercept various QuickTime VR Manager functions.

```
pascal void MyInterceptProc (
                    QTVRInstance qtvr,
                    QTVRInterceptPtr qtvrMsg,
                    SInt32 refCon,
                    Boolean *cancel);
```

qtvr	An instance of a QuickTime VR movie.
qtvrMsg	A pointer to an intercept record that specifies the function that your procedure is intercepting and the parameters for that function. See “Intercept Structure” (page 85).
refCon	The reference constant specified in the call to QTVRInstallInterceptProc that installed this intercept procedure.
cancel	On exit, a Boolean value that indicates whether the QuickTime VR Manager should not call the intercepted function when your intercept procedure returns (true) or should call the intercepted function (false).

DESCRIPTION

Your MyInterceptProc procedure is called whenever the QuickTime VR Manager is about to call the function it is intercepting (which is specified by the selector field of the intercept record pointed to by the qtvrMsg parameter). Your procedure can do any processing it deems necessary before returning.

Your intercept procedure should return, in the cancel parameter, a Boolean value that indicates whether your procedure performed the intercepted function (true) or not (false). The QuickTime VR Manager inspects that value to determine whether it should call the intercepted function after your procedure returns. Your intercept procedure can also call the QTVRCallInterceptedProc function to call the intercepted function. If your procedure does call QTVRCallInterceptedProc, then it should return the value true, unless you want the QuickTime VR Manager to call the intercepted function again.

SEE ALSO

Use QTVRInstallInterceptProc (page 125) to install an intercept procedure. See Listing 2-5 (page 50) and Listing 2-6 (page 51) for sample intercept procedures.

Node-Entering and Node-Leaving Procedures

Your application can define node-entering and node-leaving procedures that are called each time a node is entered or left.

MyEnteringNodeProc

You can define a routine to respond to a node's being entered.

```
pascal OSErr MyEnteringNodeProc (
                    QTVRInstance qtvr,
                    UInt32 nodeID,
                    SInt32 refCon);
```

`qtvr`	An instance of a QuickTime VR movie.
`nodeID`	The ID of the node being entered.
`refCon`	The reference constant specified in the call to `QTVRSetEnteringNodeProc` that installed this procedure.
function result	A result code.

DESCRIPTION

Your `MyEnteringNodeProc` procedure is called whenever a node is entered, either in response to user actions or in response to QuickTime VR Manager functions that change nodes (such as `QTVRGoToNodeID`). Your procedure can do any processing it deems necessary.

SEE ALSO

Use `QTVRSetEnteringNodeProc` (page 161) to install a node-entering procedure. See Listing 2-8 (page 54) for a sample node-entering procedure.

MyLeavingNodeProc

You can define a routine to respond to a node's being left.

```
pascal OSErr MyLeavingNodeProc (
                    QTVRInstance qtvr,
                    UInt32 fromNodeID,
                    UInt32 toNodeID,
                    Boolean *cancel,
                    SInt32 refCon);
```

`qtvr`	An instance of a QuickTime VR movie.
`fromNodeID`	The ID of the node being left.
`toNodeID`	The ID of the node to be entered, or 0 if no node is being entered (because the movie is being closed).
`cancel`	On entry, a pointer to a Boolean value. Set that value to `true` to cancel the move from `fromNodeID` to `toNodeID`; otherwise, set that value to `false`.
`refCon`	The reference constant specified in the call to `QTVRSetLeavingNodeProc` that installed this procedure.
function result	A result code.

DESCRIPTION

Your `MyLeavingNodeProc` procedure is called whenever a node is left, either in response to user actions or in response to QuickTime VR Manager functions that change nodes (such as `QTVRGoToNodeID`). Your procedure can do any processing it deems necessary.

Before returning, your procedure should set the Boolean value pointed to by the `cancel` parameter to `false` to accept the move from `fromNodeID` to `toNodeID`. Set that value to `true` to cancel the move and to remain at the node specified by the `fromNodeID` parameter.

SEE ALSO

Use `QTVRSetLeavingNodeProc` (page 162) to install a node-leaving procedure. See Listing 2-9 (page 54) for a sample node-leaving procedure.

Imaging Procedures

Your application can define prescreen and back buffer imaging procedures that are called when an image is done being drawn to the prescreen buffer used by QuickTime VR or at certain established times for a back buffer.

MyImagingCompleteProc

You can define a procedure to access QuickTime VR's prescreen buffer.

```
pascal OSErr MyImagingCompleteProc (QTVRInstance qtvr, SInt32 refCon);
```

`qtvr`	An instance of a QuickTime VR movie.
`refCon`	The reference constant specified in the call to `QTVRSetPrescreenImagingCompleteProc` that installed this procedure.
function result	A result code.

DESCRIPTION

Your `MyImagingCompleteProc` function is called whenever QuickTime VR is finished drawing an image into the prescreen buffer associated with the movie specified by the `qtvr` parameter. When your function is called, the drawing environment is set up so that you can draw directly into the current graphics world. Once your function returns, QuickTime VR copies the prescreen buffer to the final destination.

SPECIAL CONSIDERATIONS

If the value of the `kQTVRImagingDirectDraw` imaging property of the specified movie is `true`, then images are computed and drawn directly to the final destination without first being drawn into the prescreen buffer maintained by QuickTime VR. If your application has installed a prescreen buffer imaging completion procedure, QuickTime VR temporarily overrides the setting of the `kQTVRImagingDirectDraw` property and calls your `MyImagingCompleteProc` function after drawing into the prescreen buffer.

SEE ALSO

Use `QTVRSetPrescreenImagingCompleteProc` (page 176) to install a prescreen buffer imaging completion procedure. See Listing 2-10 (page 56) for a sample prescreen buffer imaging completion procedure.

MyBackBufferImagingProc

You can define a procedure to access QuickTime VR's back buffer.

```
pascal OSErr MyBackBufferImagingProc (
                    QTVRInstance qtvr,
                    Rect *drawRect,
                    UInt16 areaIndex,
                    UInt32 flagsIn,
                    UInt32 *flagsOut,
                    SInt32 refCon);
```

`qtvr` An instance of a QuickTime VR movie.

`drawRect` The rectangle, in local coordinates of the graphics world, of the area of interest.

`areaIndex` The index, in the array of area of interest structures whose address was passed to `QTVRSetBackBufferImagingProc`, of the area of interest.

`flagsIn` On entry, a set of bit flags that specify the event or operation that caused your procedure to be called, as well as other information about the state of the back buffer when your procedure is called. See "Back Buffer Imaging Procedure Flags" (page 78) for a description of the available flags.

`flagsOut` On exit, a set of bit flags that indicate what actions you performed in your procedure. See "Back Buffer Imaging Procedure Flags" (page 78) for a description of the available flags.

`refCon` The reference constant specified in the call to `QTVRSetBackBufferImagingProc` that installed this procedure.

function result A result code.

DESCRIPTION

Your `MyBackBufferImagingProc` function is called at the times specified by the `flags` parameter to the call to `QTVRSetBackBufferImagingProc` that installed that function as a back buffer imaging procedure. When your function is called, the drawing environment is set up so that you can draw directly into the current graphics world.

IMPORTANT

If the area of interest wraps around the end of the back buffer, the rectangle specified by the `drawRect` parameter is in the coordinates of an intermediate buffer that is copied into the actual back buffer when your procedure returns. ▲

You can call `QTVRRefreshBackBuffer` (page 179) in your back buffer imaging procedure to refresh the current rectangle (that is, the rectangle specified by the `drawRect` parameter).

SEE ALSO

Use `QTVRSetBackBufferImagingProc` (page 177) to install a back buffer imaging procedure.

Summary of the QuickTime VR Manager

C Summary

Constants

Version Numbers of Released APIs

```
#define kQTVRAPIMajorVersion02                  (0x02)
#define kQTVRAPIMinorVersion00                  (0x00)
#define kQTVRAPIMinorVersion01                  (0x01)
#define kQTVRAPIMinorVersion10                  (0x10)
```

Version Numbers for This API

```
#define kQTVRAPIMajorVersion kQTVRAPIMajorVersion02
#define kQTVRAPIMinorVersion kQTVRAPIMinorVersion10
```

Gestalt Selector and Response Values

```
enum {
    gestaltQTVRMgrAttr                          = FOUR_CHAR_CODE('qtvr'),
    gestaltQTVRMgrVers                          = FOUR_CHAR_CODE('qtvv'),
    gestaltQTVRMgrPresent                       = 0,
    gestaltQTVRObjMoviesPresent                 = 1,
    gestaltQTVRCylinderPanosPresent             = 2
};
```

Node Types

```
enum {
    kQTVRPanoramaType                           = FOUR_CHAR_CODE('pano'),
    kQTVRObjectType                             = FOUR_CHAR_CODE('obje')
};
```

Node IDs

```
enum {
    kQTVRCurrentNode                        = 0,
    kQTVRPreviousNode                       = 0x80000000,
    kQTVRDefaultNode                        = 0x80000001
};
```

Angular Unit Types

```
enum {
    kQTVRDegrees                            = 0,
    kQTVRRadians                            = 1
};
typedef UInt32 QTVRAngularUnits;
```

Value for flags parameter in QTVRSetPrescreenImagingCompleteProc

```
enum {
    kQTVRPreScreenEveryIdle                 = 1L << 0
};
```

Hot Spot Action Selectors

```
enum {
    kQTVRHotSpotEnter                       = 0,
    kQTVRHotSpotWithin                      = 1,
    kQTVRHotSpotLeave                       = 2
};
```

Intercept Selectors

```
enum {
    kQTVRSetPanAngleSelector                = 0x2000,
    kQTVRSetTiltAngleSelector               = 0x2001,
    kQTVRSetFieldOfViewSelector             = 0x2002,
    kQTVRSetViewCenterSelector              = 0x2003,
    kQTVRMouseEnterSelector                 = 0x2004,
    kQTVRMouseWithinSelector                = 0x2005,
    kQTVRMouseLeaveSelector                 = 0x2006,
```

```
    kQTVRMouseDownSelector                  = 0x2007,
    kQTVRMouseStillDownSelector             = 0x2008,
    kQTVRMouseUpSelector                    = 0x2009,
    kQTVRTriggerHotSpotSelector             = 0x200A,
    kQTVRGetHotSpotTypeSelector             = 0x200B
};
typedef UInt32 QTVRProcSelector;
```

Constraint Types

```
enum {
    kQTVRPan                                = 0,
    kQTVRTilt                               = 1,
    kQTVRFieldOfView                        = 2,
    kQTVRViewCenterH                        = 4,
    kQTVRViewCenterV                        = 5
};
```

Correction Modes

```
enum {
    kQTVRNoCorrection                       = 0,
    kQTVRPartialCorrection                  = 1,
    kQTVRFullCorrection                     = 2
};
```

Imaging Modes

```
enum {
    kQTVRCurrentMode                        = 0,
    kQTVRStatic                             = 1,
    kQTVRMotion                             = 2,
    kQTVRAllModes                           = 100
};
typedef UInt32 QTVRImagingMode;
```

Imaging Property Types

```
enum {
    kQTVRImagingCorrection                  = 1,
    kQTVRImagingQuality                     = 2,
```

```
    kQTVRImagingDirectDraw                  = 3,
    kQTVRImagingCurrentMode                 = 100
};

#define kQTVRImagingDefaultValue            0x80000000
```

Quality Properties

```
#define codecMinQuality                     0x000L
#define codecNormalQuality                  0x200L
#define codecMaxQuality                     0x3FFL
```

Transition Type

```
enum {
    kQTVRTransitionSwing                    = 1
};
```

Transition Properties

```
enum {
    kQTVRTransitionSpeed                    = 1,
    kQTVRTransitionDirection                = 2
};
```

Hot Spot Types

```
enum {
    kQTVRHotSpotLinkType                    = FOUR_CHAR_CODE('link'),
    kQTVRHotSpotURLType                     = FOUR_CHAR_CODE('url '),
    kQTVRHotSpotUndefinedType               = FOUR_CHAR_CODE('undf')
};
```

Interaction Property Types

```
enum {
    kQTVRInteractionMouseClickHysteresis    = 1,
    kQTVRInteractionMouseClickTimeout       = 2,
    kQTVRInteractionPanTiltSpeed            = 3,
    kQTVRInteractionZoomSpeed               = 4,
    kQTVRInteractionTranslateOnMouseDown    = 101,
```

```
    kQTVRInteractionMouseMotionScale            = 102,
    kQTVRInteractionNudgeMode                   = 103
};

#define kQTVRInteractionDefaultValue            0x80000000
```

Viewing Constraints

```
enum {
    kQTVRUnconstrained                          = 0
    kQTVRCantPanLeft                            = 1L << 0,
    kQTVRCantPanRight                           = 1L << 1,
    kQTVRCantPanUp                              = 1L << 2,
    kQTVRCantPanDown                            = 1L << 3,
    kQTVRCantZoomIn                             = 1L << 4,
    kQTVRCantZoomOut                            = 1L << 5,
    kQTVRCantTranslateLeft                      = 1L << 6,
    kQTVRCantTranslateRight                     = 1L << 7,
    kQTVRCantTranslateUp                        = 1L << 8,
    kQTVRCantTranslateDown                      = 1L << 9
};
```

Mouse Mode Values

```
enum {
    kQTVRPanning                                = 1L << 0,
    kQTVRTranslating                            = 1L << 1,
    kQTVRZooming                                = 1L << 2,
    kQTVRScrolling                              = 1L << 3,
    kQTVRSelecting                              = 1L << 4
};
```

Hot Spot Selectors

```
enum {
    kQTVRHotSpotID                              = 0,
    kQTVRHotSpotType                            = 1,
    kQTVRAllHotSpots                            = 2
};
```

Animation Settings

```
enum {
    kQTVRPalindromeViewFrames                   = 1,
    kQTVRStartFirstViewFrames                   = 2,
    kQTVRDontLoopViewFrames                     = 3,
    kQTVRPlayEveryViewFrame                     = 4,
    kQTVRSyncViewToFrameRate                    = 16,
    kQTVRPalindromeViews                        = 17
};
typedef UInt32 QTVRObjectAnimationSetting;
```

Control Settings

```
enum {
    kQTVRWrapPan                                = 1,
    kQTVRWrapTilt                               = 2,
    kQTVRCanZoom                                = 3,
    kQTVRReverseHControl                        = 4,
    kQTVRReverseVControl                        = 5,
    kQTVRSwapHVControl                          = 6,
    kQTVRTranslation                            = 7
};
typedef UInt32 QTVRControlSetting;
```

View State Types

```
enum {
    kQTVRDefault                                = 0,
    kQTVRCurrent                                = 2,
    kQTVRMouseDown                              = 3
};
typedef UInt32 QTVRViewStateType;
```

Back Buffer Imaging Procedure Flags

```
enum {
    kQTVRBackBufferEveryUpdate                  = 1L << 0,
    kQTVRBackBufferEveryIdle                    = 1L << 1,
    kQTVRBackBufferAlwaysRefresh                = 1L << 2
};
```

```
enum {
    kQTVRBackBufferRectVisible                  = 1L << 0,
    kQTVRBackBufferWasRefreshed                 = 1L << 1
};

enum {
    kQTVRBackBufferFlagDidDraw                  = 1L << 0
};
```

Nudge Directions

```
enum {
    kQTVRRight                                  = 0,
    kQTVRUpRight                                = 45,
    kQTVRUp                                     = 90,
    kQTVRUpLeft                                 = 135,
    kQTVRLeft                                   = 180,
    kQTVRDownLeft                               = 225,
    kQTVRDown                                   = 270,
    kQTVRDownRight                              = 315
};
typedef UInt32 QTVRNudgeControl;
```

Nudge Control Modes

```
enum QTVRNudgeMode {
    kQTVRNudgeRotate                            = 0,
    kQTVRNudgeTranslate                         = 1,
    kQTVRNudgeSameAsMouse                       = 2
};
typedef enum QTVRNudgeMode QTVRNudgeMode;
```

Cursor Types

```
enum {
    kQTVRUseDefaultCursor                       = 0,
    kQTVRStdCursorType                          = 1,
    kQTVRColorCursorType                        = 2
};
```

Pixel Sizes

```
enum {
    kQTVRUseMovieDepth                  = 0,
    kQTVRDepth16                        = 16,
    kQTVRDepth32                        = 32
};
```

Resolutions

```
enum {
    kQTVRDefaultRes                     = 0,
    kQTVRFullRes                        = 1L << 0,
    kQTVRHalfRes                        = 1L << 1,
    kQTVRQuarterRes                     = 1L << 2
};
```

Geometry Constants

```
enum {
    kQTVRUseMovieGeometry               = 0,
    kQTVRVerticalCylinder               = 'vcyl'
};
```

Cache Sizes

```
enum {
    kQTVRMinimumCache                   = -1,
    kQTVRSuggestedCache                 = 0,
    kQTVRFullCache                      = 1
};
```

Data Types

QuickTime VR Movie Instances

```
typedef struct QTVRRecord               *QTVRInstance;
```

Intercept Structure

```
typedef struct QTVRInterceptRecord {
    SInt32                              reserved1;
    SInt32                              selector;
    SInt32                              reserved2;
    SInt32                              reserved3;
    SInt32                              paramCount;
    void                                *parameter[6];
} QTVRInterceptRecord, *QTVRInterceptPtr;
```

Floating-Point Point Structure

```
struct QTVRFloatPoint {
    float                       x;
    float                       y;
};
typedef struct QTVRFloatPoint QTVRFloatPoint;
```

Cursor Record

```
struct QTVRCursorRecord {
    UInt16                      theType;            /* field was previously named
                                                       "type"*/
    SInt16                      rsrcID;
    Handle                      handle;
};
typedef struct QTVRCursorRecord QTVRCursorRecord;
```

Area of Interest Structure

```
struct QTVRAreaOfInterest {
    float                       panAngle;
    float                       tiltAngle;
    float                       width;
    float                       height;
    UInt32                      flags;
};
typedef struct QTVRAreaOfInterest QTVRAreaOfInterest;
```

QuickTime VR Manager Routines

Initializing and Terminating QuickTime VR

```
OSErr InitializeQTVR            ( void );

OSErr TerminateQTVR             ( void );
```

Initializing and Managing QuickTime VR Movie Instances

```
Track QTVRGetQTVRTrack          (Movie theMovie, SInt32 index);

OSErr QTVRGetQTVRInstance       (QTVRInstance *qtvr,
                                 Track qtvrTrack,
                                 MovieController mc);
```

Manipulating Viewing Angles and Zooming

```
float QTVRGetPanAngle           (QTVRInstance qtvr);

OSErr QTVRSetPanAngle           (QTVRInstance qtvr, float panAngle);

float QTVRGetTiltAngle          (QTVRInstance qtvr);

OSErr QTVRSetTiltAngle          (QTVRInstance qtvr, float tiltAngle);

float QTVRGetFieldOfView        (QTVRInstance qtvr);

OSErr QTVRSetFieldOfView        (QTVRInstance qtvr, float fieldOfView);

OSErr QTVRGetViewCenter         (QTVRInstance qtvr, QTVRFloatPoint *viewCenter);

OSErr QTVRSetViewCenter         (QTVRInstance qtvr, const QTVRFloatPoint
                                 *viewCenter);

OSErr QTVRNudge                 (QTVRInstance qtvr, QTVRNudgeControl direction);

OSErr QTVRInteractionNudge      (QTVRInstance qtvr, QTVRNudgeControl direction);

OSErr QTVRShowDefaultView       (QTVRInstance qtvr);
```

Getting Scene and Node Information

```
OSErr QTVRGetVRWorld            (QTVRInstance qtvr, QTAtomContainer *VRWorld);

OSErr QTVRGoToNodeID            (QTVRInstance qtvr, UInt32 nodeID);

UInt32 QTVRGetCurrentNodeID     (QTVRInstance qtvr);
```

```
OSType QTVRGetNodeType            (QTVRInstance qtvr, UInt32 nodeID);
OSErr QTVRGetNodeInfo             (QTVRInstance qtvr,
                                   UInt32 nodeID,
                                   QTAtomContainer *nodeInfo);
```

Managing Hot Spots

```
OSErr QTVRPtToHotSpotID           (QTVRInstance qtvr, Point pt, UInt32 *hotSpotID);
OSErr QTVRGetHotSpotType          (QTVRInstance qtvr,
                                   UInt32 hotSpotID,
                                   OSType *hotSpotType);
OSErr QTVRTriggerHotSpot          (QTVRInstance qtvr,
                                   UInt32 hotSpotID,
                                   QTAtomContainer nodeInfo,
                                   QTAtom selectedAtom);
OSErr QTVREnableHotSpot           (QTVRInstance qtvr,
                                   UInt32 enableFlag,
                                   UInt32 hotSpotValue,
                                   Boolean enable);
OSErr QTVRSetMouseOverHotSpotProc (QTVRInstance qtvr,
                                   QTVRMouseOverHotSpotUPP mouseOverHotSpotProc,
                                   SInt32 refCon,
                                   UInt32 flags);
UInt32 QTVRGetVisibleHotSpots     (QTVRInstance qtvr, Handle hotSpots);
OSErr QTVRGetHotSpotRegion        (QTVRInstance qtvr,
                                   UInt32 hotSpotID,
                                   RgnHandle hotSpotRegion);
```

Handling Events

```
Boolean QTVRGetMouseOverTracking  (QTVRInstance qtvr);
OSErr QTVRSetMouseOverTracking    (QTVRInstance qtvr, Boolean enable);
OSErr QTVRMouseEnter              (QTVRInstance qtvr, Point pt, UInt32 *hotSpotID);
OSErr QTVRMouseWithin             (QTVRInstance qtvr, Point pt, UInt32 *hotSpotID);
OSErr QTVRMouseLeave              (QTVRInstance qtvr, Point pt);
Boolean QTVRGetMouseDownTracking  (QTVRInstance qtvr);
```

```
OSErr QTVRSetMouseDownTracking    (QTVRInstance qtvr, Boolean enable);

OSErr QTVRMouseDown               (QTVRInstance qtvr,
                                   Point pt,
                                   UInt32 when,
                                   UInt16 modifiers,
                                   UInt32 *hotSpotID);

OSErr QTVRMouseStillDown          (QTVRInstance qtvr, Point pt, UInt32 *hotSpotID);

OSErr QTVRMouseStillDownExtended  (QTVRInstance qtvr,
                                   Point pt,
                                   UInt32 *hotSpotID,
                                   WindowPtr w,
                                   UInt32 when,
                                   UInt16 modifiers);

OSErr QTVRMouseUp                 (QTVRInstance qtvr, Point pt, UInt32 *hotSpotID);

OSErr QTVRMouseUpExtended         (QTVRInstance qtvr,
                                   Point pt,
                                   UInt32 *hotSpotID,
                                   WindowPtr w,
                                   UInt32 when,
                                   UInt16 modifiers);
```

Intercepting QuickTime VR Manager Routines

```
OSErr QTVRInstallInterceptProc    (QTVRInstance qtvr,
                                   QTVRProcSelector selector,
                                   QTVRInterceptUPP interceptProc,
                                   SInt32 refCon,
                                   UInt32 flags);

OSErr QTVRCallInterceptedProc     (QTVRInstance qtvr, QTVRInterceptRecord *qtvrMsg);
```

Managing Object Nodes

```
UInt32 QTVRGetCurrentMouseMode    (QTVRInstance qtvr);

float QTVRGetFrameRate            (QTVRInstance qtvr);

OSErr QTVRSetFrameRate            (QTVRInstance qtvr, float rate);

float QTVRGetViewRate             (QTVRInstance qtvr);

OSErr QTVRSetViewRate             (QTVRInstance qtvr, float rate);
```

```
TimeValue QTVRGetCurrentViewDuration (
                                    QTVRInstance qtvr);
TimeValue QTVRGetViewCurrentTime  (QTVRInstance qtvr);
OSErr QTVRSetViewCurrentTime      (QTVRInstance qtvr, TimeValue time);
UInt16 QTVRGetViewStateCount      (QTVRInstance qtvr);
OSErr QTVRGetViewState            (QTVRInstance qtvr,
                                    QTVRViewStateType viewStateType,
                                    UInt16 *state);
OSErr QTVRSetViewState            (QTVRInstance qtvr,
                                    QTVRViewStateType viewStateType,
                                    UInt16 state);
OSErr QTVRGetAnimationSetting     (QTVRInstance qtvr,
                                    QTVRObjectAnimationSetting setting,
                                    Boolean *enable);
OSErr QTVRSetAnimationSetting     (QTVRInstance qtvr,
                                    QTVRObjectAnimationSetting setting,
                                    Boolean enable);
OSErr QTVRGetControlSetting       (QTVRInstance qtvr,
                                    QTVRControlSetting setting,
                                    Boolean *enable);
OSErr QTVRSetControlSetting       (QTVRInstance qtvr,
                                    QTVRControlSetting setting,
                                    Boolean enable);
Boolean QTVRGetFrameAnimation     (QTVRInstance qtvr);
OSErr QTVREnableFrameAnimation    (QTVRInstance qtvr, Boolean enable);
Boolean QTVRGetViewAnimation      (QTVRInstance qtvr);
OSErr QTVREnableViewAnimation     (QTVRInstance qtvr, Boolean enable);
```

Managing Imaging Characteristics

```
Boolean QTVRGetVisible            (QTVRInstance qtvr);
OSErr QTVRSetVisible              (QTVRInstance qtvr, Boolean visible);
```

```
OSErr QTVRGetImagingProperty      (QTVRInstance qtvr,
                                   QTVRImagingMode imagingMode,
                                   UInt32 imagingProperty,
                                   SInt32 *propertyValue);
OSErr QTVRSetImagingProperty      (QTVRInstance qtvr,
                                   QTVRImagingMode imagingMode,
                                   UInt32 imagingProperty,
                                   SInt32 propertyValue);
OSErr QTVRUpdate                  (QTVRInstance qtvr, QTVRImagingMode imagingMode);
OSErr QTVRBeginUpdateStream       (QTVRInstance qtvr, QTVRImagingMode imagingMode);
OSErr QTVREndUpdateStream         (QTVRInstance qtvr);
OSErr QTVRSetTransitionProperty   (QTVRInstance qtvr,
                                   UInt32 transitionType,
                                   UInt32 transitionProperty,
                                   SInt32 transitionValue);
OSErr QTVREnableTransition        (QTVRInstance qtvr,
                                   UInt32 transitionType,
                                   Boolean enable);
```

Converting Angles and Points

```
QTVRAngularUnits QTVRGetAngularUnits (
                                   QTVRInstance qtvr);
OSErr QTVRSetAngularUnits         (QTVRInstance qtvr, QTVRAngularUnits units);
OSErr QTVRPtToAngles              (QTVRInstance qtvr,
                                   Point pt,
                                   float *panAngle,
                                   float *tiltAngle);
OSErr QTVRCoordToAngles           (QTVRInstance qtvr,
                                   QTVRFloatPoint *coord,
                                   float *panAngle,
                                   float *tiltAngle);
OSErr QTVRAnglesToCoord           (QTVRInstance qtvr,
                                   float panAngle,
                                   float tiltAngle,
                                   QTVRFloatPoint *coord);
short QTVRPanToColumn             (QTVRInstance qtvr, float panAngle);
```

```
float QTVRColumnToPan            (QTVRInstance qtvr, short column);
short QTVRTiltToRow              (QTVRInstance qtvr, float tiltAngle);
float QTVRRowToTilt              (QTVRInstance qtvr, short row);
OSErr QTVRWrapAndConstrain       (QTVRInstance qtvr,
                                  short kind,
                                  float value,
                                  float *result);
```

Managing QuickTime VR Movie Interaction

```
OSErr QTVRSetEnteringNodeProc    (QTVRInstance qtvr,
                                  QTVREnteringNodeUPP enteringNodeProc,
                                  SInt32 refCon,
                                  UInt32 flags);
OSErr QTVRSetLeavingNodeProc     (QTVRInstance qtvr,
                                  QTVRLeavingNodeUPP leavingNodeProc,
                                  SInt32 refCon,
                                  UInt32 flags);
OSErr QTVRGetInteractionProperty (QTVRInstance qtvr,
                                  UInt32 property,
                                  void *value);
OSErr QTVRSetInteractionProperty (QTVRInstance qtvr,
                                  UInt32 property,
                                  void *value);
OSErr QTVRReplaceCursor          (QTVRInstance qtvr, QTVRCursorRecord *cursRecord);
```

Determining Viewing Limits and Constraints

```
OSErr QTVRGetViewingLimits       (QTVRInstance qtvr,
                                  UInt16 kind,
                                  float *minValue,
                                  float *maxValue);
UInt32 QTVRGetConstraintStatus   (QTVRInstance qtvr);
OSErr QTVRGetConstraints         (QTVRInstance qtvr,
                                  UInt16 kind,
                                  float *minValue,
                                  float *maxValue);
```

```
OSErr QTVRSetConstraints          (QTVRInstance qtvr,
                                   UInt16 kind,
                                   float minValue,
                                   float maxValue);
```

Managing Memory

```
OSErr QTVRGetAvailableResolutions (QTVRInstance qtvr, UInt16 *resolutionsMask);

OSErr QTVRGetBackBufferMemInfo    (QTVRInstance qtvr,
                                   UInt16 geometry,
                                   UInt16 resolution,
                                   UInt32 cachePixelFormat,
                                   SInt32 *minCacheBytes,
                                   SInt32 *suggestedCacheBytes,
                                   SInt32 *fullCacheBytes);

OSErr QTVRGetBackBufferSettings   (QTVRInstance qtvr,
                                   UInt16 *geometry,
                                   UInt16 *resolution,
                                   UInt32 *cachePixelFormat,
                                   SInt16 *cacheSize);

OSErr QTVRSetBackBufferPrefs      (QTVRInstance qtvr,
                                   UInt16 geometry,
                                   UInt16 resolution,
                                   UInt32 cachePixelFormat,
                                   SInt16 cacheSize);
```

Accessing Image Buffers

```
OSErr QTVRSetPrescreenImagingCompleteProc (
                                   QTVRInstance qtvr,
                                   ImagingCompleteUPP imagingCompleteProc,
                                   SInt32 refCon,
                                   UInt32 flags);

OSErr QTVRSetBackBufferImagingProc(QTVRInstance qtvr,
                                   BackBufferImagingUPP backBufferImagingProc,
                                   UInt16 numAreas,
                                   QTVRAreaOfInterest *areasOfInterest,
                                   SInt32 refCon);

OSErr QTVRRefreshBackBuffer       (QTVRInstance qtvr, UInt32 flags);
```

Application-Defined Routines

Mouse Over Hot Spot Procedure

```
pascal OSErr MyMouseOverHotSpotProc (
                                    QTVRInstance qtvr,
                                    UInt32 hotSpotID,
                                    UInt32 flags,
                                    SInt32 refCon);
```

QuickTime VR Intercept Routine

```
pascal void MyInterceptProc       (QTVRInstance qtvr,
                                    QTVRInterceptPtr qtvrMsg,
                                    SInt32 refCon,
                                    Boolean *cancel);
```

Node-Entering and Node-Leaving Procedures

```
pascal OSErr MyEnteringNodeProc   (QTVRInstance qtvr, UInt32 nodeID, SInt32 refCon);

pascal OSErr MyLeavingNodeProc    (QTVRInstance qtvr,
                                    UInt32 fromNodeID,
                                    UInt32 toNodeID,
                                    Boolean *cancel,
                                    SInt32 refCon);
```

Imaging Procedures

```
pascal OSErr MyImagingCompleteProc(QTVRInstance qtvr, SInt32 refCon);

pascal OSErr MyBackBufferImagingProc (
                                    QTVRInstance qtvr,
                                    Rect *drawRect,
                                    UInt16 areaIndex,
                                    UInt32 flagsIn,
                                    UInt32 *flagsOut,
                                    SInt32 refCon);
```

Result Codes

notAQTVRMovieErr	–30540	The specified movie isn't a QuickTime VR movie
constraintReachedErr	–30541	A view constraint has been reached
callNotSupportedByNodeErr	–30542	The specified function isn't supported by this node
selectorNotSupportedByNodeErr	–30543	The specified selector isn't supported by this node
invalidNodeIDErr	–30544	The specified node ID is invalid
invalidViewStateErr	–30545	The specified view state is invalid
timeNotInViewErr	–30546	The specified time is not in the view
propertyNotSupportedByNodeErr	–30547	The specified property is not supported by this node
settingNotSupportedByNodeErr	–30548	The specified setting is not supported by this node
qtvrLibraryLoadErr	–30554	Unable to find or load QTVR library
qtvrUninitialized	–30555	QuickTime VR Manager has not been initialized

CHAPTER 3

QuickTime VR Movie Controller

Contents

QuickTime VR Movie Controller

This chapter describes the QuickTime VR movie controller, a movie controller component that manages the interface for presenting QuickTime VR movies to users and allowing them to navigate and explore in those movies. You can use standard QuickTime movie controller functions to configure and manipulate the QuickTime VR movie controller.

You need to read this chapter if you want to customize the interface presented by the QuickTime VR movie controller (for example, to hide the controller bar). You might also need to read this chapter to learn how the QuickTime VR movie controller handles movie controller actions. Your application can issue actions to access certain movie controller capabilities; your application can also install an action filter function to intercept and possibly also override movie controller actions.

This chapter begins by describing the appearance and behavior of the QuickTime VR movie controller. Then it describes the movie controller actions and the ways in which your application might need to issue or respond to them. The section "Using the QuickTime VR Movie Controller," beginning on page 213, briefly illustrates how to issue a movie controller action and perform other operations on the QuickTime VR movie controller.

The section "QuickTime VR Movie Controller Reference" (page 217), provides a complete reference of the ways in which the QuickTime VR movie controller handles movie controller actions. The section "Summary of the QuickTime VR Movie Controller" (page 235), summarizes the currently defined movie controller actions and movie control flags that are defined in the header file `Movies.h`, which is part of QuickTime. The summary also lists the movie controller actions that are specific to QuickTime VR (and that are defined in the header file `QuickTimeVR.h`).

Note
For complete information on movie controllers, see the chapter "Movie Controller Components" in the book *Inside Macintosh: QuickTime Components*. You need to be familiar with the information in that chapter in order to use this chapter. ◆

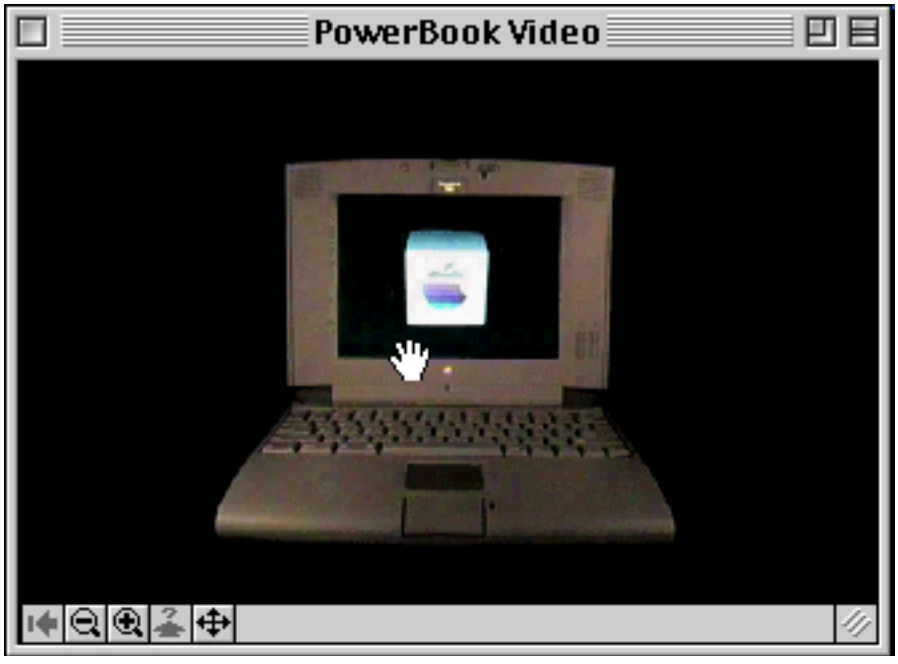
About the QuickTime VR Movie Controller

The **QuickTime VR movie controller** is a movie controller component that manages the interface for presenting QuickTime VR movies to users and allowing them to navigate and explore in those movies. This component is stored in the QuickTime VR extension and is loaded automatically whenever an application calls `NewMovieController` with a QuickTime VR movie. A special piece of user data in a QuickTime VR movie file indicates the movie controller to use; see “QuickTime VR Movie Creation” (page 267) for instructions on including that user data in a QuickTime VR file.

Elements of the QuickTime VR Movie Controller

The QuickTime VR movie controller provides control elements for zooming in and out and several other navigational controls. Figure 3-1 shows the elements supported by the QuickTime VR movie controller component.

Figure 3-1 The QuickTime VR movie controller



The user can navigate in a movie by holding the mouse button down and dragging inside the picture. The user can also use the **controller bar** to perform several other operations. The controller bar contains the following controls:

- A **go-back button.** This control allows the user to return to the previous node. Clicking this button restores the previous static pan angle, tilt angle, and field of view. This button is enabled only for multinode movies.
- A **zoom-out button.** This control allows the user to zoom out. Holding down the mouse button while the cursor is over this control causes the field of view of the displayed node to increase, thereby making the object or panorama appear to move away from the viewer.
- A **zoom-in button.** This control allows the user to zoom in. Holding down the mouse button while the cursor is over this control causes the field of view of the displayed node to decrease, thereby making the object or panorama appear to move toward the viewer.

- A **hot spot display button.** This control allows the user to highlight the visible hot spots. Holding down the mouse button while the cursor is over this control causes any hot spots in the currently visible portion of the panorama or object to be highlighted (a border is drawn around each hot spot, and the enclosed region is filled in a distinctive color). Releasing the mouse button restores the image to its unhighlighted state.
  Double-clicking the mouse button causes the hot spot display to latch in the on state so that the hot spots remain displayed during manipulation of the VR movie. In that state, a single mouse click turns the hot spot display off. This button is disabled for movies that do not have hot spots.
- A **translate mode button.** This control allows the user to enable or disable translate mode. When translate mode is enabled, dragging the mouse causes an object to be translated instead of panned or tilted. This control is enabled only for object nodes.

In addition to these buttons, the controller bar also contains a **label display area** in which helpful information is displayed. For instance, when the cursor is over one of the buttons, the button's name appears in the label display area. Similarly, when the cursor is over a hot spot, the hot spot's name (if it has one) appears in the label display area.

You can programmatically change which of these buttons are displayed by issuing the appropriate movie controller actions; see "Movie Control Flags" (page 233) for details. In addition, you can hide the controller bar entirely; see "Hiding and Showing the Controller Bar" (page 213) for details.

Movie Controller Actions

A **movie controller action** is a constant that you can pass to a movie controller to request that the movie controller perform some action (such as modify certain movie characteristics or respond to user events). For example, you can pass the `mcActionSetCursorSettingEnabled` action to enable or disable the automatic cursor tracking and shape changing provided by the QuickTime VR movie controller.

There are two ways in which you might be concerned with these actions: your application can invoke these actions directly by calling the `MCDoAction` function; or your application can install an **action filter function,** which can receive any of these actions; your action filter can then either intercept the action or send it back to the movie controller for processing.

A movie controller action is usually accompanied by some parameter data. For instance, the `mcActionSetCursorSettingEnabled` action must be accompanied by a Boolean value that indicates whether to enable or disable cursor tracking and shape changing. When calling `MCDoAction`, you get or set data through the `params` parameter. Similarly, an action filter function exchanges data with a movie controller through its `params` parameter. The type and meaning of this additional parameter data are described in the individual descriptions of each movie controller action. See "Movie Controller Actions" (page 217).

Note
For complete information on handling movie controller actions, see the chapter "Movie Controller Components" in the book *Inside Macintosh: QuickTime Components.* ◆

Using the QuickTime VR Movie Controller

This section illustrates basic ways of interacting with the QuickTime VR movie controller. In particular, it provides source code examples that show how you can

- hide the controller bar
- hide and show buttons in the controller bar
- disable the automatic cursor tracking and shape changing provided by the QuickTime VR movie controller

Note
The code examples shown in this section provide only very rudimentary error handling. ◆

Hiding and Showing the Controller Bar

You can use standard QuickTime movie component routines to hide and show the controller bar associated with a QuickTime VR movie. To hide the controller bar, you can call the `MCSetVisible` function, as illustrated in Listing 3-1.

Listing 3-1 Hiding the controller bar

```
componentResult     myResult;
Boolean             isVisible;

isVisible = false;
myResult = MCSetVisible(myMC, isVisible);
```

Showing and Hiding Controller Bar Buttons

You can use standard QuickTime movie controller routines to hide and show specific buttons in the controller bar associated with a QuickTime VR movie. The QuickTime VR movie controller automatically shows and hides some buttons, and it automatically disables some buttons that might not be appropriate for a specific movie or node. You can, however, override these automatic behaviors using the QuickTime VR Manager.

For instance, the QuickTime VR movie controller displays the speaker button (used for adjusting a movie's volume) whenever a movie contains a sound track. It's possible, however, that only a single node in a large multinode movie has a sound track. In that case, you might want to hide the speaker button in all nodes that do not have a sound track. Conversely, the QuickTime VR movie controller hides the speaker button if a movie does not contain a sound track. You might, however, play a sound loaded from a sound resource or from another QuickTime file. In that case, you might want to show the speaker button and have it control the sound you're playing. In both these cases, you need to override the default behavior of the QuickTime VR movie controller.

Note first that every VR movie has *two* sets of movie controller flags: a set of control flags and a set of explicit flags. The control flags work as described in "Movie Control Flags" and documented in *Inside Macintosh: QuickTime Components:* If a bit in the set of control flags is set (that is, equal to 1), then the associated action or property is enabled. For instance, bit 17 (`mcFlagQTVRSuppressZoomBtns`) means to suppress the zoom buttons. So, if that bit is set in a VR movie's control flags, the zoom buttons are not displayed. If that bit is clear, the zoom buttons are displayed.

However, the QuickTime VR movie controller sometimes suppresses buttons even when those buttons have not been explicitly suppressed in the control flags. As already mentioned, if a particular VR movie does not contain a sound track, then the movie controller automatically suppresses the speaker button. If

a movie does contain a sound track, then the speaker button is displayed only if the suppress speaker bit is off.

This might not be what you'd like to happen. For instance, if your application is playing a sound that it loaded from a sound resource, you might want the user to be able to adjust the sound's volume using the volume control. To do that, you need a way to force the speaker button to appear. For this reason, the explicit flags were introduced.

The explicit flags indicate which bits in the control flags are to be used explicitly (that is, taken at face value). If a certain bit is set in a movie's explicit flags, then the corresponding bit in the control flags is interpreted as the desired setting for the feature (and the movie controller does not attempt to do anything clever). In other words, if bit 17 is set in a movie's explicit flags and bit 17 is clear in that movie's control flags, then the zoom buttons are always displayed. Similarly, if bit 2 is set in a movie's explicit flags and bit 2 is clear in that movie's control flags, then the speaker button is displayed, whether or not the movie contains a sound track.

To get or set a bit in a movie's explicit flags, you must set the flag `mcFlagQTVRExplicitFlagSet` in your call to `mcActionGetFlags` or `mcActionSetFlags`. To get or set a bit in a movie's control flags, you must clear the flag `mcFlagQTVRExplicitFlagSet` in your call to `mcActionGetFlags` or `mcActionSetFlags`. Note that when you use the defined constants to set values in the explicit flags, the constant names might be confusing. For instance, setting the bit `mcFlagSuppressSpeakerButton` in a movie's explicit flags doesn't cause the speaker to be suppressed; it just means: "use the actual value of the `mcFlagSuppressSpeakerButton` bit in the control flags."

Now you can see how to hide or show a button in the controller bar: set the appropriate explicit flag to 1 and set the corresponding control flag to the desired value. Listing 3-2 shows how to force a specific button in the controller bar to be displayed.

Listing 3-2 Showing a controller bar button

```
void ShowControllerButton (MovieController theMC, long theButton)
{
    long    myControllerFlags;

    // Get the current explicit flags
    // and set the explicit flag for the specified button.
```

```
    myControllerFlags = mcFlagQTVRExplicitFlagSet;
    MCDoAction(theMC, mcActionGetFlags, &myControllerFlags);
    MCDoAction(theMC, mcActionSetFlags,
            (void *)((myControllerFlags | theButton) | mcFlagQTVRExplicitFlagSet));

    // Get the current control flags
    // and clear the suppress flag for the specified button.
    myControllerFlags = 0;
    MCDoAction(theMC, mcActionGetFlags, &myControllerFlags);
    MCDoAction(theMC, mcActionSetFlags,
            (void *)(myControllerFlags & ~theButton));
}
```

Listing 3-3 shows how to force a specific button in the controller bar to be hidden. Because the suppress flag overrides the setting of the explicit flag, this routine sets only the suppress flag and doesn't bother with the explicit flag.

Listing 3-3 Hiding a controller bar button

```
void HideControllerButton (MovieController theMC, long theButton)
{
    long    myControllerFlags;

    // Get the current control flags
    // and set the suppress flag for the specified button.
    myControllerFlags = 0;
    MCDoAction(theMC, mcActionGetFlags, &myControllerFlags);
    MCDoAction(theMC, mcActionSetFlags,
            (void *)((myControllerFlags | theButton));
}
```

Sending Actions to the QuickTime VR Movie Controller

You can use the `MCDoAction` function to send a movie controller action to a movie controller. For example, you can execute this line of code to disable the automatic cursor tracking and shape changing provided by the QuickTime VR movie controller:

```
MCDoAction(myMC, mcActionSetCursorSettingEnabled, (void*) false);
```

In this example, the `myMC` parameter is an identifier for the QuickTime VR movie controller, returned by a previous call to `NewMovieController`.

QuickTime VR Movie Controller Reference

This section provides a complete reference for the QuickTime VR movie controller. It describes how the QuickTime VR movie controller handles the currently defined movie controller actions. It also describes the meanings of the movie control flags that can be retrieved from or passed to the QuickTime VR movie controller using the `mcActionGetFlags` and `mcActionSetFlags` movie controller actions.

Constants

This section describes the constants associated with the QuickTime VR movie controller.

Movie Controller Actions

Movie controller actions are defined by these constants:

```
enum {
    mcActionIdle                                = 1,
    mcActionDraw                                = 2,
    mcActionActivate                            = 3,
    mcActionDeactivate                          = 4,
    mcActionMouseDown                           = 5,
    mcActionKey                                 = 6,
    mcActionPlay                                = 8,
    mcActionGoToTime                            = 12,
    mcActionSetVolume                           = 14,
    mcActionGetVolume                           = 15,
    mcActionStep                                = 18,
    mcActionSetLooping                          = 21,
    mcActionGetLooping                          = 22,
    mcActionSetLoopIsPalindrome                 = 23,
    mcActionGetLoopIsPalindrome                 = 24,
```

```
    mcActionSetGrowBoxBounds                    = 25,
    mcActionControllerSizeChanged               = 26,
    mcActionSetSelectionBegin                   = 29,
    mcActionSetSelectionDuration                = 30,
    mcActionSetKeysEnabled                      = 32,
    mcActionGetKeysEnabled                      = 33,
    mcActionSetPlaySelection                    = 34,
    mcActionGetPlaySelection                    = 35,
    mcActionSetUseBadge                         = 36,
    mcActionGetUseBadge                         = 37,
    mcActionSetFlags                            = 38,
    mcActionGetFlags                            = 39,
    mcActionSetPlayEveryFrame                   = 40,
    mcActionGetPlayEveryFrame                   = 41,
    mcActionGetPlayRate                         = 42,
    mcActionShowBalloon                         = 43,
    mcActionBadgeClick                          = 44,
    mcActionMovieClick                          = 45,
    mcActionSuspend                             = 46,
    mcActionResume                              = 47,
    mcActionSetControllerKeysEnabled            = 48,
    mcActionGetTimeSliderRect                   = 49,
    mcActionMovieEdited                         = 50,
    mcActionGetDragEnabled                      = 51,
    mcActionSetDragEnabled                      = 52,
    mcActionGetSelectionBegin                   = 53,
    mcActionGetSelectionDuration                = 54,
    mcActionPrerollAndPlay                      = 55,
    mcActionGetCursorSettingEnabled             = 56,
    mcActionSetCursorSettingEnabled             = 57,
    mcActionSetColorTable                       = 58,
    mcActionLinkToURL                           = 59,
    mcActionCustomButtonClick                   = 60,
    mcActionForceTimeTableUpdate                = 61,
    mcActionSetControllerTimeLimits             = 62,
    mcActionExecuteAllActionsForQTEvent         = 63,
    mcActionExecuteOneActionForQTEvent          = 64,
    mcActionAdjustCursor                        = 65,
    mcActionUseTrackForTimeTable                = 66,
    mcActionClickAndHoldPoint                   = 67,
```

```
    mcActionShowMessageString                       = 68
                                /* param is a StringPtr*/
};
```

The action descriptions that follow are divided into those that can be used by your application and those that can be received by your action filter function. Many actions fall into both categories.

Actions for use by applications

`mcActionIdle` Your application can use this action to grant event-processing time to a movie controller. There is no parameter for this action.

`mcActionDraw` Your application can use this action to send an update event to a movie controller.

The parameter for this action is a pointer to a window.

`mcActionActivate` Your application can use this action to activate a movie controller. There is no parameter for this action.

`mcActionDeactivate` Your application can use this action to deactivate a movie controller. There is no parameter for this action.

`mcActionMouseDown` Your application can use this action to pass a mouse-down event to a movie controller.

The parameter data must contain a pointer to an event structure (of type `EventRecord`); the `message` field in the event structure must specify the window in which the user clicked.

`mcActionKey` Your application can use this action to pass a key-down or auto-key event to a movie controller.

The parameter data must contain a pointer to an event structure that describes the key event.

`mcActionPlay` Your application can use this action to start or stop playing a frame animation. For objects with no frame animation or for panoramas, this action is ignored.

The parameter data must contain a floating-point value that indicates the frame rate. Values greater than 0.0 correspond to forward rates; values less than 0.0 play the animation backward. A value of 0.0 stops the animation.

`mcActionGoToTime` Your application can use this action to move to a specific view time in an animated object movie. For objects with no frame animation or for panoramas, this action is ignored.

The parameter data must contain a pointer to a time structure that specifies the target position in the movie.

`mcActionSetVolume` Your application can use this action to set a movie's volume.

The parameter data must contain a pointer to a 16-bit fixed-point number that indicates the relative volume of the movie. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

`mcActionGetVolume` Your application can use this action to determine a movie's volume.

The parameter data must contain a pointer to a 16-bit fixed-point number that indicates the relative volume of the movie. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

`mcActionStep` This action is not supported by the QuickTime VR movie controller.

`mcActionSetLooping` Your application can use this action to enable or disable frame animation looping for a movie. For objects with no frame animation or for panoramas, this action is ignored.

The parameter data must contain a Boolean value; a value of `true` indicates that looping is to be enabled.

`mcActionGetLooping` Your application can use this action to determine whether frame animation for an object movie is looping.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if frame animation looping is enabled for the object movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetLoopIsPalindrome`

Your application can use this action to enable palindrome frame animation looping. Looping must also be enabled for palindrome looping to take effect.

The parameter data must contain a Boolean value; a value of `true` indicates that palindrome looping is to be enabled.

`mcActionGetLoopIsPalindrome`

Your application can use this action to determine whether palindrome frame animation looping is enabled for a movie. Looping must also be enabled for palindrome looping to take effect.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if palindrome looping is enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetGrowBoxBounds`

Your application can use this action to set the limits for resizing a movie.

The parameter data must contain a pointer to a rectangle; set the rectangle to the boundary coordinates for the movie. If you want to prevent the movie from being resized, supply an empty rectangle (note that enabling or disabling the size box may change the appearance of some movie controllers). By default, movie controllers do not have size boxes. You must use this action to establish a size box for a movie controller.

If the movie controller's boundary rectangle intersects the lower-right corner of your window, your window cannot have a size box.

`mcActionSetSelectionBegin`

This action is not supported by the QuickTime VR movie controller.

`mcActionSetSelectionDuration`

This action is not supported by the QuickTime VR movie controller.

`mcActionSetKeysEnabled`

Your application can use this action to enable or disable keystrokes for a movie.

The parameter data must contain a Boolean value; a value of `true` indicates that keystrokes are to be enabled. By default, this value is set to `false`.

`mcActionGetKeysEnabled`

Your application can use this action to determine whether keystrokes are enabled for a movie controller.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if keystrokes are enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetPlaySelection`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetPlaySelection`

This action is not supported by the QuickTime VR movie controller.

`mcActionSetUseBadge`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetUseBadge`

This action is not supported by the QuickTime VR movie controller.

`mcActionSetFlags` Your application can use this action to set a movie's control flags.

The parameter data must contain a long integer that contains the new control flag values. See "Movie Control Flags" (page 233) for a description of the available flags.

`mcActionGetFlags` Your application can use this action to retrieve a movie's control flags.

The parameter data must contain a pointer to a long integer. The movie controller places the movie's control flags into that long integer. See "Movie Control Flags" (page 233) for a description of the available flags.

`mcActionSetPlayEveryFrame`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetPlayEveryFrame`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetPlayRate`

Your application can use this action to determine an object movie's frame rate. You set the frame rate by using the `mcActionPlay` action.

The parameter data must contain a pointer to a floating-point value. The movie controller returns the movie's frame rate in that value. Values greater than 0.0 correspond to forward rates; values less than 0.0 play the movie backward. A value of 0.0 indicates that the frame animation is stopped.

`mcActionBadgeClick` Your application can use this action to indicate that the badge was clicked. The parameter is a pointer to a Boolean value. On entry, the Boolean value is set to `true`. Set the Boolean value to `false` if you want the controller to ignore the click in the badge.

`mcActionMovieClick` Your application can use this action to indicate that the movie was clicked. The parameter is a pointer to an event structure containing the mouse-down event. If you want the controller to ignore the mouse-down event, change the `what` field of the event structure to a null event.

`mcActionSuspend` Your application can use this action to indicate that a suspend event has been received. There is no parameter for this action.

`mcActionResume` Your application can use this action to indicate that a resume event has been received. There is no parameter for this action.

`mcActionSetControllerKeysEnabled`

Your application can use this action to control whether the controller bar keys are enabled or disabled.

The parameter data must contain a Boolean value; a value of `true` indicates that the controller bar keys are to be enabled. By default, this value is set to `true`.

`mcActionGetTimeSliderRect`

This action is not supported by the QuickTime VR movie controller.

`mcActionMovieEdited`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetDragEnabled`

Your application can use this action to determine whether dragging and dropping is enabled.

The parameter data must contain a pointer to a Boolean value. The QuickTime VR movie controller always sets this value to `false`.

`mcActionSetDragEnabled`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetSelectionBegin`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetSelectionDuration`

This action is not supported by the QuickTime VR movie controller.

`mcActionPrerollAndPlay`

Your application can use this action to preroll a movie and then immediately play it using view animation.

The parameter data must contain a floating-point value that indicates the view rate. Values greater than 0.0 correspond to forward rates; values less than 0.0 play the movie backward. A value of 0.0 indicates that the view animation is stopped.

`mcActionGetCursorSettingEnabled`

Your application can use this action to get the current state of cursor tracking and shape changing.

The parameter data must contain a pointer to a Boolean value; the movie controller sets this value to `true` if the controller has been instructed to track the cursor and change its shape as appropriate. Otherwise, the controller sets the value to `false`.

`mcActionSetCursorSettingEnabled`

Your application can use this action to enable or disable the automatic cursor tracking and shape changing provided by the QuickTime VR movie controller.

The parameter data must contain a Boolean value; a value of `true` indicates that the cursor tracking and shape changing are to be enabled.

`mcActionSetColorTable`

This action is not supported by the QuickTime VR movie controller.

`mcActionLinkToURL`

The parameter is a handle to URL; it can be sent to the controller or intercepted when generated by a `'url '` hotspot.

`mcActionAdjustCursor`

The parameter is a pointer to `EventRecord` (`WindowPtr` is in message parameter). The action is not supported in QuickTime VR 2.1, but will be supported in future versions.

`mcActionClickAndHoldPoint`

The parameter is a point (local coordinates). The action returns true if the point has click and hold action (for example, a QuickTime VR object movie autorotate spot).

`mcActionShowMessageString`

The parameter is a `StringPtr`. The action is not supported by the QuickTime VR movie controller at this time, but will likely be used in future versions to allow the application to set the string shown in the control bar.

Actions for use by action filter functions

`mcActionIdle` Your action filter function receives this action when the application has granted null event-processing time to the movie controller. There is no parameter for this action.

`mcActionDraw` Your filter function receives this action when the controller has received an update event.

The parameter for this action is a pointer to a window.

`mcActionActivate` Your filter function receives this action when the controller has received an activate or resume event. There is no parameter for this action.

`mcActionDeactivate` Your filter function receives this action when the controller has received a deactivate or suspend event. There is no parameter for this action.

`mcActionMouseDown` Your action filter function receives this action when the movie controller has received a mouse-down event.

The parameter data must contain a pointer to an event structure; the `message` field in the event structure must specify the window in which the user clicked.

`mcActionKey` Your action filter function receives this action when the movie controller has received a key-down or auto-key event.

	The parameter data must contain a pointer to an event structure that describes the key event.
`mcActionPlay`	Your action filter function receives this action when the movie controller has received a request to start or stop playing a frame animation. The parameter data must contain a floating-point value that indicates the frame rate. Values greater than 0.0 correspond to forward rates; values less than 0.0 play the animation backward. A value of 0.0 stops the animation.
`mcActionGoToTime`	Your action filter function receives this action when the movie controller has received a request to go to a specified view time in an animated object movie. The parameter data must contain a pointer to a time structure that specifies the target position in the movie.
`mcActionSetVolume`	Your action filter function receives this action when the movie controller has received a request to set the movie's volume. The parameter data must contain a pointer to a 16-bit fixed-point number that indicates the relative volume of the movie. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.
`mcActionGetVolume`	Your action filter function receives this action when the movie controller has received a request to retrieve the movie's volume. The parameter data must contain a pointer to a 16-bit fixed-point number that indicates the relative volume of the movie. Volume values range from –1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.
`mcActionStep`	This action is not supported by the QuickTime VR movie controller.
`mcActionSetLooping`	Your action filter function receives this action when the movie controller has received a request to turn frame animation looping on or off. The parameter data must contain a Boolean value; a value of `true` indicates that looping is to be enabled.

`mcActionGetLooping` Your action filter function receives this action when the controller has received a request to indicate whether frame animation looping is enabled for its movie.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if frame animation looping is enabled for the object movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetLoopIsPalindrome`

Your action filter function receives this action when the movie controller has received a request to turn palindrome frame animation looping on or off. Looping must also be enabled for palindrome looping to take effect.

The parameter data must contain a Boolean value; a value of `true` indicates that palindrome looping is to be enabled.

`mcActionGetLoopIsPalindrome`

Your action filter function receives this action when the controller has received a request to indicate whether palindrome frame animation looping is enabled for its movie.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if palindrome looping is enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetGrowBoxBounds`

Your action filter function receives this action when the movie controller has received a request to set the limits for resizing the movie.

The parameter data contains a pointer to a rectangle; the rectangle defines the boundary coordinates for the movie. If the rectangle is empty, the application wants to disable the size box. You may change the appearance of your controller in response to such a request.

`mcActionControllerSizeChanged`

Your filter function receives this action when the user has resized the movie controller; the controller component issues this action before it updates the screen, allowing

your application to change the controller's location or appearance before the user sees the resized controller.

There is no parameter for this action.

An application should never use this action.

`mcActionSetSelectionBegin`

This action is not supported by the QuickTime VR movie controller.

`mcActionSetSelectionDuration`

This action is not supported by the QuickTime VR movie controller.

The parameter data must contain a pointer to a time structure specifying the ending time of the movie's current selection.

`mcActionSetKeysEnabled`

Your action filter function receives this action when the movie controller has received a request to enable or disable keystrokes.

The parameter data must contain a Boolean value; a value of `true` indicates that keystrokes are to be enabled. By default, this value is set to `false`.

`mcActionGetKeysEnabled`

Your filter function receives this action when the controller has received a request to indicate whether keystrokes are enabled for its movie.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if keystrokes are enabled for the movie that is assigned to this controller. Otherwise, it sets the value to `false`.

`mcActionSetPlaySelection`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetPlaySelection`

This action is not supported by the QuickTime VR movie controller.

`mcActionSetUseBadge`

Your action filter function receives this action when the movie controller has received a request to turn the badge on or off.

The parameter data must contain a Boolean value; a value of `true` indicates that the badge is to be enabled.

`mcActionGetUseBadge`

Your action filter function receives this action when the controller has received a request to indicate whether it is using a badge.

The parameter data must contain a pointer to a Boolean value. The movie controller sets this value to `true` if the controller is using a badge. Otherwise, it sets the value to `false`.

`mcActionSetFlags` Your action filter function receives this action when the movie controller has received a request to set the movie's control flags.

The parameter data must contain a long integer that contains the new control flag values. See "Movie Control Flags" (page 233) for a description of the available flags.

`mcActionGetFlags` Your action filter function receives this action when the movie controller has received a request to retrieve the movie's control flags.

The parameter data must contain a pointer to a long integer. The movie controller places the movie's control flags into that long integer. See "Movie Control Flags" (page 233) for a description of the available flags.

`mcActionSetPlayEveryFrame`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetPlayEveryFrame`

This action is not supported by the QuickTime VR movie controller.

`mcActionShowBalloon`

Your action filter function receives this action when the controller wants to display Balloon Help. Your filter function instructs the controller whether to display the Balloon Help. This action allows you to override the movie controller's default Balloon Help behavior.

The parameter data contains a pointer to a Boolean value. Set the value to `true` to display the appropriate Balloon Help. Otherwise, set the value to `false`.

An application should never use this action.

`mcActionBadgeClick` Your action filter function receives this action when the badge is clicked. The parameter is a pointer to a Boolean value. On entry, the Boolean value is set to `true`. Set the Boolean value to `false` if you want the controller to ignore the click in the badge.

`mcActionMovieClick` Your action filter function receives this action when the movie is clicked. The parameter is a pointer to an event structure containing the mouse-down event. If you want the controller to ignore the mouse-down event, change the `what` field of the event structure to a null event.

`mcActionSuspend` Your action filter function receives this action when a suspend event has been received. There is no parameter for this action.

`mcActionResume` Your action filter function receives this action when a resume event has been received. There is no parameter for this action.

`mcActionSetControllerKeysEnabled`

Your action filter function receives this action when the movie controller has received a request to enable or disable the controller bar keys.

The parameter data must contain a Boolean value; a value of `true` indicates that the controller bar keys are to be enabled. By default, this value is set to `true`.

`mcActionGetTimeSliderRect`

This action is not supported by the QuickTime VR movie controller.

`mcActionMovieEdited`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetDragEnabled`

Your action filter function receives this action when the controller has received a request to indicate whether dragging and dropping is enabled.

The parameter data must contain a pointer to a Boolean value. The QuickTime VR movie controller always sets this value to `false`.

`mcActionSetDragEnabled`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetSelectionBegin`

This action is not supported by the QuickTime VR movie controller.

`mcActionGetSelectionDuration`

This action is not supported by the QuickTime VR movie controller.

`mcActionPrerollAndPlay`

Your action filter function receives this action when the movie controller has received a request to preroll a movie and then immediately play it using view animation.

The parameter data must contain a floating-point value that indicates the view rate. Values greater than 0.0 correspond to forward rates; values less than 0.0 play the movie backward. A value of 0.0 indicates that the view animation is stopped.

`mcActionGetCursorSettingEnabled`

Your action filter function receives this action when the movie controller has received a request to get the current state of cursor tracking and shape changing.

The parameter data must contain a pointer to a Boolean value; the movie controller sets this value to `true` if the controller has been instructed to track the cursor and change its shape as appropriate. Otherwise, the controller sets the value to `false`.

`mcActionSetCursorSettingEnabled`

Your action filter function receives this action when the movie controller has received a request to enable or disable the automatic cursor tracking and shape changing provided by the QuickTime VR movie controller.

The parameter data must contain a Boolean value; a value of `true` indicates that the cursor tracking and shape changing are to be enabled.

`mcActionSetColorTable`

This action is not supported by the QuickTime VR movie controller.

`mcActionLinkToURL`

The parameter is a Handle to URL; it can be sent to the controller or intercepted when generated by a `'url '` hotspot.

`mcActionCustomButtonClick`

The parameter is a pointer to `EventRecord`, where in local coordinates. Generated (intercept only) when the custom button is clicked; see `mcFlagsUseCustomButton` (page 234).

`mcActionForceTimeTableUpdate`

This action is not supported by the QuickTime VR movie controller.

`mcActionSetControllerTimeLimits`

Not for use by application.

`mcActionExecuteAllActionsForQTEvent`

This action is not supported by the QuickTime VR movie controller.

`mcActionExecuteOneActionForQTEvent`

This action is not supported by the QuickTime VR movie controller.

`mcActionAdjustCursor`

The parameter is a pointer to `EventRecord` (`WindowPtr` is in message parameter). Not supported in QuickTime VR 2.1, but will be supported in future versions.

`mcActionUseTrackForTimeTable`

Not for use by application.

`mcActionClickAndHoldPoint`

The parameter is a point (local coordinates). The action returns true if the point has click and hold action (for example, a QuickTime VR object movie autorotate spot).

`mcActionShowMessageString`

The parameter is a `StringPtr`. Not supported by the QuickTime VR movie controller at this time, but will likely be used in future versions to allow the application to set the string shown in the control bar.

Movie Control Flags

The `mcActionGetFlags` and `mcActionSetFlags` movie controller actions instruct a movie controller to get and set the movie's control flags, which are encoded into a long integer. You can use these constants to read or write the bits in that long integer:

```
enum {
    mcFlagSuppressMovieFrame                = 1 << 0,
    mcFlagSuppressStepButtons               = 1 << 1,
    mcFlagSuppressSpeakerButton             = 1 << 2,
    mcFlagsUseWindowPalette                 = 1 << 3,
    mcFlagsDontInvalidate                   = 1 << 4,
    mcFlagsUseCustomButton                  = 1 << 5,
    mcFlagQTVRSuppressBackBtn               = 1 << 16,
    mcFlagQTVRSuppressZoomBtns              = 1 << 17,
    mcFlagQTVRSuppressHotSpotBtn            = 1 << 18,
    mcFlagQTVRSuppressTranslateBtn          = 1 << 19,
    mcFlagQTVRSuppressHelpText              = 1 << 20,
    mcFlagQTVRSuppressHotSpotNames          = 1 << 21
};
```

Constant descriptions

`mcFlagSuppressMovieFrame`
: This flag is not supported by the QuickTime VR movie controller.

`mcFlagSuppressStepButtons`
: This flag is not supported by the QuickTime VR movie controller.

`mcFlagSuppressSpeakerButton`
: Controls whether the controller displays the speaker button. If this flag is set to 1, the controller does not display the speaker button. By default, this flag is set to 0.

`mcFlagsUseWindowPalette`
: Controls whether the controller manages the palette for the window containing the movie. This ensures that a movie's colors are reproduced as accurately as possible. This flag is particularly useful for movies with custom color tables. If this flag is set to 1, the movie controller does not manage the window palette. By default, this flag is set to 0.

`mcFlagsDontInvalidate`

Controls whether the controller boundary rectangle is invalidated (and hence updated when an update event occurs). If this flag is set to 1, the controller does not invalidate the controller boundary rectangle. By default, this flag is set to 0.

`mcFlagsUseCustomButton`

Causes a button to be displayed for application-specific use. A click on this button generates an `mcActionCustomButtonClick`.

`mcFlagQTVRSuppressBackBtn`

Controls whether the controller displays the go-back button. If this flag is set to 1, the controller does not display the go-back button. By default, this flag is set to 0.

`mcFlagQTVRSuppressZoomBtns`

Controls whether the controller displays the zoom-in and zoom-out buttons. If this flag is set to 1, the controller does not display the zoom buttons. By default, this flag is set to 0.

`mcFlagQTVRSuppressHotSpotBtn`

Controls whether the controller displays the hot spot display button. If this flag is set to 1, the controller does not display the hot spot display button. By default, this flag is set to 0.

`mcFlagQTVRSuppressTranslateBtn`

Controls whether the controller displays the translate mode button. If this flag is set to 1, the controller does not display the translate mode button. By default, this flag is set to 0.

`mcFlagQTVRSuppressHelpText`

Controls whether the controller displays help text in the label display area. If this flag is set to 1, the controller does not display help text. By default, this flag is set to 0.

`mcFlagQTVRSuppressHotSpotNames`

Controls whether the controller displays hot spot names in the label display area. If this flag is set to 1, the controller does not display hot spot names. By default, this flag is set to 0.

Summary of the QuickTime VR Movie Controller

C Summary

Constants

Movie Controller Actions

```
enum {
    mcActionIdle                                = 1,
    mcActionDraw                                = 2,
    mcActionActivate                            = 3,
    mcActionDeactivate                          = 4,
    mcActionMouseDown                           = 5,
    mcActionKey                                 = 6,
    mcActionPlay                                = 8,
    mcActionGoToTime                            = 12,
    mcActionSetVolume                           = 14,
    mcActionGetVolume                           = 15,
    mcActionStep                                = 18,
    mcActionSetLooping                          = 21,
    mcActionGetLooping                          = 22,
    mcActionSetLoopIsPalindrome                 = 23,
    mcActionGetLoopIsPalindrome                 = 24,
    mcActionSetGrowBoxBounds                    = 25,
    mcActionControllerSizeChanged               = 26,
    mcActionSetSelectionBegin                   = 29,
    mcActionSetSelectionDuration                = 30,
    mcActionSetKeysEnabled                      = 32,
    mcActionGetKeysEnabled                      = 33,
    mcActionSetPlaySelection                    = 34,
    mcActionGetPlaySelection                    = 35,
    mcActionSetUseBadge                         = 36,
    mcActionGetUseBadge                         = 37,
```

```
    mcActionSetFlags                        = 38,
    mcActionGetFlags                        = 39,
    mcActionSetPlayEveryFrame               = 40,
    mcActionGetPlayEveryFrame               = 41,
    mcActionGetPlayRate                     = 42,
    mcActionShowBalloon                     = 43,
    mcActionBadgeClick                      = 44,
    mcActionMovieClick                      = 45,
    mcActionSuspend                         = 46,
    mcActionResume                          = 47,
    mcActionSetControllerKeysEnabled        = 48,
    mcActionGetTimeSliderRect               = 49,
    mcActionMovieEdited                     = 50,
    mcActionGetDragEnabled                  = 51,
    mcActionSetDragEnabled                  = 52,
    mcActionGetSelectionBegin               = 53,
    mcActionGetSelectionDuration            = 54,
    mcActionPrerollAndPlay                  = 55,
    mcActionGetCursorSettingEnabled         = 56,
    mcActionSetCursorSettingEnabled         = 57,
    mcActionSetColorTable                   = 58
};
```

Movie Control Flags

```
enum {
    mcFlagSuppressMovieFrame                = 1 << 0,
    mcFlagSuppressStepButtons               = 1 << 1,
    mcFlagSuppressSpeakerButton             = 1 << 2,
    mcFlagsUseWindowPalette                 = 1 << 3,
    mcFlagsDontInvalidate                   = 1 << 4,
    mcFlagQTVRSuppressBackBtn               = 1 << 16,
    mcFlagQTVRSuppressZoomBtns              = 1 << 17,
    mcFlagQTVRSuppressHotSpotBtn            = 1 << 18,
    mcFlagQTVRSuppressTranslateBtn          = 1 << 19,
    mcFlagQTVRSuppressHelpText              = 1 << 20,
    mcFlagQTVRSuppressHotSpotNames          = 1 << 21
};
```

Data Types

```
typedef short                                   mcAction;

typedef unsigned long                           MCFlags;
```

CHAPTER 4

QuickTime VR Atom Containers

Contents

This chapter describes in detail the VR world and node information atom containers. These two atom containers can be obtained by calling the QuickTime VR Manager routines `QTVRGetVRWorld` and `QTVRGetNodeInfo`. Those routines are described in Chapter 2, "QuickTime VR Manager.".

You need to know about the various atoms contained in the VR world and node information atom containers if you want to extract information from a QuickTime VR file that cannot be obtained using VR Manager functions. For instance, there is no QuickTime VR Manager function that returns the name of a given node; however, you can easily get a node's name by reading the information in the atoms in the atom container returned by the `QTVRGetNodeInfo` function.

In addition, if you want your application to be able to create QuickTime VR movies, you to need to know about these atom containers as well as the information contained in Chapter 5, "Creating QuickTime VR Movies."

Note
In general, you don't need to know about the format of atoms or atom containers simply to use the functions provided by the QuickTime VR Manager. ◆

About Atom Containers

A QuickTime atom container is a basic structure for storing information in QuickTime files. An atom container is a tree structured hierarchy of QT atoms. By definition, only the **leaf atoms** in the hierarchy contain data. Intermediate atoms serve as **parent atoms** that contain any number of child atoms, which may in turn be either leaf atoms or more parent atoms. Each parent's child atom is uniquely identified by its **atom type** and **atom ID.** The atom container itself is considered the parent of the highest level atoms.

Many of the atom types contained in the VR world and node information atom containers are unique within their container. For example, each has a single header atom. Most of the parent atoms within an atom container are unique as well, such as the node parent atom in the VR world atom container or the hot spot parent atom in the node information atom container. For these one time only atoms, the atom ID is always set to 1. Unless otherwise mentioned in the descriptions of the atoms that follow, assume that the atom ID is 1.

Many of the atom structures contain two version fields, `majorVersion` and `minorVersion`. The values of these fields correspond to the constants `kQTVRMajorVersion` and `kQTVRMinorVersion` found in the header file `QuickTimeVRFormat.h`. For QuickTime 2.0 files, these values are 2 and 0.

QuickTime provides many routines for creating and accessing atom containers. Those are described in *QuickTime 3 Reference* .

The String Atom and the String Encoding Atom

Some of the leaf atoms within the VR world and node information atom containers contain fields that specify the ID of **string atoms** that are siblings of the leaf atom. For example, the VR world header atom contains a field for the name of the scene. The string atom is a leaf atom whose atom type is `kQTVRStringAtomType ('vrsg')`. Its atom ID is that specified by the referring leaf atom.

A string atom contains a string. The structure of a string atom is defined by the `VRStringAtom` data type:

```
typedef struct VRStringAtom {
    UInt16                          stringUsage;
    UInt16                          stringLength;
    unsigned char                   theString[4];
} VRStringAtom, *VRStringAtomPtr;
```

Field descriptions

`stringUsage`	The string usage. Currently, this field is unused.
`stringLength`	The length, in bytes, of the string.
`theString`	The string. The string atom structure is extended to hold this string.

Each string atom may also have a sibling leaf atom called the **string encoding atom.** The string encoding atom's atom type is `kQTVRStringEncodingAtomType ('vrse')`. Its atom ID is the same as that of the corresponding string atom. The string encoding atom contains a single variable, `TextEncoding`, a `UInt32`, as defined in the header file `TextCommon.h`. The value of `TextEncoding` is handed, along with the string, to the routine `QTTextToNativeText` for conversion for display on the current machine. The routine `QTTextToNativeText` is found in the header file `Movies.h`.

Note
The header file `TextCommon.h` contains constants and routines for generating and handling text encodings. ◆

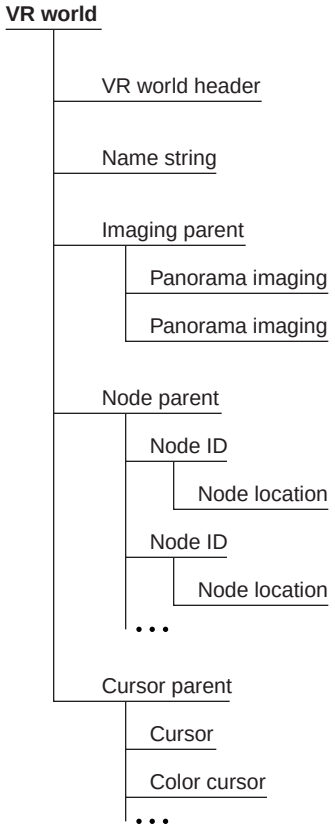
VR World Atom Container

The **VR world atom container** (VR world for short) includes such information as the name for the entire scene, the default node ID, and default imaging properties, as well as a list of the nodes contained in the QTVR track.

A VR world can also contain custom scene information. QuickTime VR ignores any atom types that it doesn't recognize, but you can extract those atoms from the VR world using standard QuickTime atom functions.

The structure of the VR world atom container is shown in Figure 4-1. The component atoms are defined and their structures are shown in the sections that follow.

Figure 4-1 Structure of the VR world atom container



The following sections describe the atom types found in a VR world atom container:

- "VR World Header Atom Structure"
- "Imaging Parent Atom"
- "Panorama-Imaging Atom"
- "Node Parent Atom"
- "Node Location Atom Structure"
- "Custom Cursor Atoms"

VR World Header Atom Structure

The **VR world header atom** is a leaf atom. Its atom type is `kQTVRWorldHeaderAtomType` (`'vrsc'`). It contains the name of the scene and the default node ID to be used when the file is first opened as well as fields reserved for future use.

The structure of a VR world header atom is defined by the `VRWorldHeaderAtom` data type:

```
typedef struct VRWorldHeaderAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    QTAtomID                            nameAtomID;
    UInt32                              defaultNodeID;
    UInt32                              vrWorldFlags;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRWorldHeaderAtom, *VRWorldHeaderAtomPtr;
```

Field descriptions

`majorVersion`	The major version number of the file format.
`minorVersion`	The minor version number of the file format.
`nameAtomID`	The ID of the string atom that contains the name of the scene. That atom should be a sibling of the VR world header atom. The value of this field is 0 if no name string atom exists.
`defaultNodeID`	The ID of the default node (that is, the node to be displayed when the file is first opened).
`vrWorldFlags`	A set of flags for the VR world. Currently, this field is unused.
`reserved1`	Reserved. This field must be 0.
`reserved2`	Reserved. This field must be 0.

Imaging Parent Atom

The **imaging parent atom** is the parent atom of one or more node-specific imaging atoms. Its atom type is `kQTVRImagingParentAtomType ('imgp')`. Currently only panoramas have an imaging atom defined.

Panorama-Imaging Atom

A **panorama-imaging atom** describes the default imaging characteristics for all the panoramic nodes in a scene. This atom overrides QuickTime VR's own defaults. Those defaults are described in the section "QTVRSetImagingProperty" (page 146).

The panorama-imaging atom has an atom type of `kQTVRPanoImagingAtomType` (`'impn'`). Generally, there is one panorama-imaging atom for each imaging mode (see below), so the atom ID, while it must be unique for each atom, is ignored. QuickTime VR iterates through all the panorama-imaging atoms.

The structure of a panorama-imaging atom is defined by the `VRPanoImagingAtom` data type:

```
typedef struct VRPanoImagingAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    UInt32                              imagingMode;
    UInt32                              imagingValidFlags;
    UInt32                              correction;
    UInt32                              quality;
    UInt32                              directDraw;
    UInt32                              imagingProperties[6];
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRPanoImagingAtom, *VRPanoImagingAtomPtr;
```

Field descriptions

`majorVersion`	The major version number of the file format.
`minorVersion`	The minor version number of the file format.
`imagingMode`	The imaging mode to which the default values apply. Only `kQTVRStatic` and `kQTVRMotion` are allowed here. See "Imaging Modes" (page 63) for a description of the available imaging modes.
`imagingValidFlags`	A set of flags that indicate which imaging property fields in this structure are valid. See "Imaging Property Valid Flags" (page 247) for a description.
`correction`	The default correction mode for panoramic nodes. This can be either `kQTVRNoCorrection`, `kQTVRPartialCorrection`, or `kQTVRFullCorrection`. See "Correction Modes" (page 62) for a description.

`quality`	The default imaging quality for panoramic nodes. See “Quality Properties” (page 65) for a description of the imaging qualities.
`directDraw`	The default direct-drawing property for panoramic nodes. This can be `true` or `false`. See “Imaging Property Types” (page 64) for a description of the direct-drawing property values.
`imagingProperties`	Reserved for future panorama-imaging properties.
`reserved1`	Reserved. This field must be 0.
`reserved2`	Reserved. This field must be 0.

Imaging Property Valid Flags

The `imagingValidFlags` field in the panorama-imaging atom structure specifies which imaging property fields in that structure are valid. You can use these bit flags to specify a value for that field:

```
enum {
    kQTVRValidCorrection                    = 1 << 0,
    kQTVRValidQuality                       = 1 << 1,
    kQTVRValidDirectDraw                    = 1 << 2,
    kQTVRValidFirstExtraProperty            = 1 << 3
};
```

Constant descriptions

`kQTVRValidCorrection`
: If this bit is set, the `correction` field holds a default correction mode.

`kQTVRValidQuality`
: If this bit is set, the `quality` field holds a default imaging quality.

`kQTVRValidDirectDraw`
: If this bit is set, the `directDraw` field holds a default direct-drawing property.

`kQTVRValidFirstExtraProperty`
: If this bit is set, the first element in the array in the `imagingProperties` field holds a default imaging property.

Node Parent Atom

The **node parent atom** is the parent of one or more **node ID atoms**. The atom type of the node parent atom is `kQTVRNodeParentAtomType` (`'vrnp'`) and the atom type of the each node ID atom is `kQTVRNodeIDAtomType` (`'vrni'`). There is one node ID atom for each node in the file. The atom ID of the node ID atom is the node ID of the node. The node ID atom is the parent of the node location atom. Currently the node location atom is the only child atom defined for the node ID atom. Its atom type is `kQTVRNodeLocationAtomType ('nloc')`.

Node Location Atom Structure

Currently the node location atom is the only child atom defined for the node ID atom. Its atom type is `kQTVRNodeLocationAtomType ('nloc')`. A node location atom describes the type of a node and its location.

The structure of a node location atom is defined by the `VRNodeLocationAtom` data type:

```
typedef struct VRNodeLocationAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    OSType                              nodeType;
    UInt32                              locationFlags;
    UInt32                              locationData;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRNodeLocationAtom, *VRNodeLocationAtomPtr;
```

Field descriptions

`majorVersion` The major version number of the file format.

`minorVersion` The minor version number of the file format.

`nodeType` The node type. See “Node Types” (page 58) for a description of the available node types. Currently, this field should contain either `kQTVRPanoramaType` or `kQTVRObjectType`.

`locationFlags` The location flags. Currently, this field must contain the value `kQTVRSameFile`, indicating that the node is to be found in the current file. In future, these flags may indicate that the node is in a different file or at some URL location.

`locationData`	The location of the node data. When the `locationFlags` field is `kQTVRSameFile`, this field should be 0. The nodes are found in the file in the same order that they are found in the node list.
`reserved1`	Reserved. This field must be 0.
`reserved2`	Reserved. This field must be 0.

Custom Cursor Atoms

As described in section "Hot Spot Information Atom" (page 252), the hot spot information atom allows you to indicate custom cursor IDs for particular hot spots that replace the default cursors used by QuickTime VR. QuickTime VR allows you to store your custom cursors in the VR world of the movie file.

Note
If you're using the Mac OS, you could store your custom cursors in the resource fork of the movie file. However, this would not work on any other platform (such as Windows 95), so storing cursors in the resource fork of the movie file is not recommended. ◆

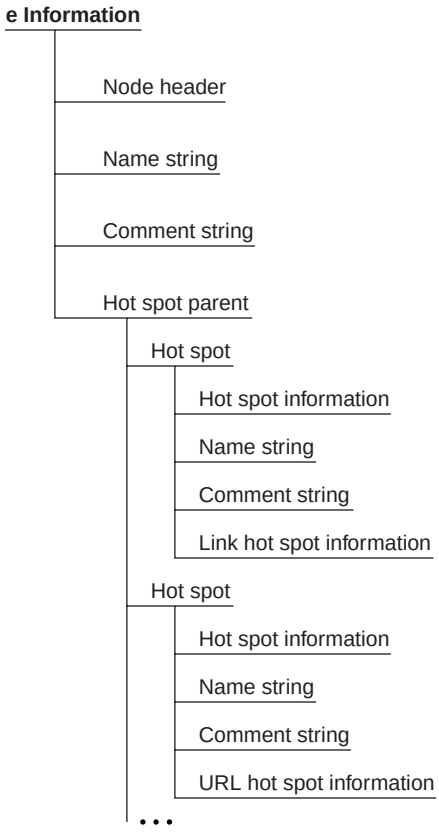
The **cursor parent atom** is the parent of all of the custom cursor atoms stored in the VR world. Its atom type is `kQTVRCursorParentAtomType` (`'vrcp'`). The child atoms of the cursor parent are either cursor atoms or color cursor atoms. Their atom types are `kQTVRCursorAtomType` (`'CURS'`) and `kQTVRColorCursorAtomType` (`'crsr'`). These atoms are stored exactly as cursors or color cursors would be stored as a resource. See the Cursor Utilities chapter of *Inside Macintosh: Imaging With QuickDraw* for details.

Node Information Atom Container

The **node information atom container** includes general information about the node such as the node's type, ID, and name. The node information atom container also contains the list of hot spot atoms for the node. A QuickTime VR movie contains one node information atom container for each node in the file. The routine `QTVRGetNodeInfo` allows you to obtain the node information atom container for the current node or for any other node in the movie.

Figure 4-2 shows the structure of the node information atom container.

Figure 4-2 Structure of the node information atom container



Node Header Atom Structure

A **node header atom** is a leaf atom that describes the type and ID of a node, as well as other information about the node. Its atom type is `kQTVRNodeHeaderAtomType` (`'ndhd'`).

The structure of a node header atom is defined by the `VRNodeHeaderAtom` data type:

```
typedef struct VRNodeHeaderAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    OSType                              nodeType;
    QTAtomID                            nodeID;
    QTAtomID                            nameAtomID;
    QTAtomID                            commentAtomID;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRNodeHeaderAtom, *VRNodeHeaderAtomPtr;
```

Field descriptions

`majorVersion`	The major version number of the file format.
`minorVersion`	The minor version number of the file format.
`nodeType`	The node type. See “Node Types” (page 58) for a description of the available node types. Currently, this field should contain either `kQTVRPanoramaType` or `kQTVRObjectType`.
`nodeID`	The node ID.
`nameAtomID`	The ID of the string atom that contains the name of the node. This atom should be a sibling of the node header atom. The value of this field is 0 if no name string atom exists.
`commentAtomID`	The ID of the string atom that contains a comment for the node. This atom should be a sibling of the node header atom. The value of this field is 0 if no comment string atom exists.
`reserved1`	Reserved. This field must be 0.
`reserved2`	Reserved. This field must be 0.

Hot Spot Parent Atom

The **hot spot parent atom** is the parent for all **hot spot atoms** for the node. The atom type of the hot spot parent atom is `kQTVRHotSpotParentAtomType` (`'hspa'`) and the atom type of the each hot spot atom is `kQTVRHotSpotAtomType` (`'hots'`). The atom ID of each hot spot atom is the hot spot ID for the corresponding hot spot. The hot spot ID is determined by its color index value as it is stored in the hot spot image track. (See Chapter 5, “Creating QuickTime VR Movies.”)

Each hot spot atom is the parent of a number of atoms that contain information about each hot spot.

Hot Spot Information Atom

The **hot spot information atom** contains general information about a hot spot. Its atom type is `kQTVRHotSpotInfoAtomType` (`'hsin'`). Every hot spot atom should have a hot spot information atom as a child.

The structure of a hot spot information atom is defined by the `VRHotSpotInfoAtom` data type:

```
typedef struct VRHotSpotInfoAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    OSType                              hotSpotType;
    QTAtomID                            nameAtomID;
    QTAtomID                            commentAtomID;
    SInt32                              cursorID[3];
    Float32                             bestPan;
    Float32                             bestTilt;
    Float32                             bestFOV;
    FloatPoint                          bestViewCenter;
    Rect                                hotSpotRect;
    UInt32                              flags;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRHotSpotInfoAtom, *VRHotSpotInfoAtomPtr;
```

Field descriptions

`majorVersion` The major version number of the file format.

`minorVersion` The minor version number of the file format.

`hotSpotType` The hot spot type. This type specifies which other information atoms—if any—are siblings to this one. QuickTime VR recognizes three types: `kQTVRHotSpotLinkType`, `kQTVRHotSpotURLType`, and `kQTVRHotSpotUndefinedType`.

`nameAtomID` The ID of the string atom that contains the name of the hot spot. This atom should be a sibling of the hot spot

	information atom. This string is displayed in the QuickTime VR controller bar when the mouse is moved over the hot spot.
`commentAtomID`	The ID of the string atom that contains a comment for the hot spot. This atom should be a sibling of the hot spot information atom. The value of this field is 0 if no comment string atom exists.
`cursorID`	An array of three IDs for custom hot spot cursors (that is, cursors that override the default hot spot cursors provided by QuickTime VR). The first ID (`cursorID[0]`) specifies the cursor that is displayed when it is in the hot spot. The second ID (`cursorID[1]`) specifies the cursor that is displayed when it is in the hot spot and the mouse button is down. The third ID (`cursorID[2]`) specifies the cursor that is displayed when it is in the hot spot and the mouse button is released. To retain the default cursor for any of these operations, set the corresponding cursor ID to 0. Custom cursors should be stored in the VR world atom container, as described in “VR World Atom Container” (page 243).
`bestPan`	The best pan angle for viewing this hot spot.
`bestTilt`	The best tilt angle for viewing this hot spot.
`bestFOV`	The best field of view for viewing this hot spot.
`bestViewCenter`	The best view center for viewing this hot spot; applies only to object nodes.
`hotSpotRect`	The boundary box for this hot spot, specified as the number of pixels in full panoramic space. This field is valid only for panoramic nodes.
`flags`	A set of hot spot flags. Currently, this field is unused.
`reserved1`	Reserved. This field must be 0.
`reserved2`	Reserved. This field must be 0.

Note
In QuickTime VR movie files, all angular values are stored as 32-bit floating-point values that specify degrees. In addition, all floating-point values conform to the IEEE Standard 754 for binary floating-point arithmetic, in big-endian format. ◆

Specific Information Atoms

Depending on the value of the `hotSpotType` field in the hot spot info atom there may also be a type specific information atom. The atom type of the type specific atom is the hot spot type. For information, please see “Hot Spot Types” (page 67) in Chapter 2, “QuickTime VR Manager.”

Link Hot Spot Atom

The **link hot spot atom** specifies information for hot spots of type `kQTVRHotSpotLinkType ('link')`. Its atom type is thus `'link'`. The link hot spot atom contains specific information about a link hot spot.

The structure of a link hot spot atom is defined by the `VRLinkHotSpotAtom` data type:

```
typedef struct VRLinkHotSpotAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    UInt32                              toNodeID;
    UInt32                              fromValidFlags;
    Float32                             fromPan;
    Float32                             fromTilt;
    Float32                             fromFOV;
    FloatPoint                          fromViewCenter;
    UInt32                              toValidFlags;
    Float32                             toPan;
    Float32                             toTilt;
    Float32                             toFOV;
    FloatPoint                          toViewCenter;
    Float32                             distance;
    UInt32                              flags;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRLinkHotSpotAtom, *VRLinkHotSpotAtomPtr;
```

Field descriptions

`majorVersion` The major version number of the file format.

`minorVersion` The minor version number of the file format.

`toNodeID`	The ID of the destination node (that is, the node to which this hot spot is linked).
`fromValidFlags`	A set of flags that indicate which source node view settings are valid. See “Link Hot Spot Valid Flags” (page 255) for a description of the available flags.
`fromPan`	The preferred from-pan angle at the source node (that is, the node containing the hot spot).
`fromTilt`	The preferred from-tilt angle at the source node.
`fromFOV`	The preferred from-field of view at the source node.
`fromViewCenter`	The preferred from-view center at the source node.
`toValidFlags`	A set of flags that indicate which destination node view settings are valid. See “Link Hot Spot Valid Flags” (page 255) for a description of the available flags.
`toPan`	The pan angle to use when displaying the destination node.
`toTilt`	The tilt angle to use when displaying the destination node.
`toFOV`	The field of view to use when displaying the destination node.
`toViewCenter`	The view center to use when displaying the destination node.
`distance`	The distance between the source node and the destination node.
`flags`	A set of link hot spot flags. Currently, this field is unused and should be set to 0.
`reserved1`	Reserved. This field must be 0.
`reserved2`	Reserved. This field must be 0.

Certain fields in the link hot spot atom are not currently used by QuickTime VR. The `fromValidFlags` field is generally set to zero and the `from` fields are not used. However, these fields could be quite useful if you have created a transition movie from one node to another. The from angles can be used to swing the current view of the source node to align with the first frame of the transition movie. The `distance` field is intended for use with 3D applications, but is also not currently used by QuickTime VR.

Link Hot Spot Valid Flags

The `toValidFlags` field in the link hot spot atom structure specifies which view settings are to be used when moving to a destination node from a hot spot. You can use these bit flags to specify a value for that field:

```
enum {
    kQTVRValidPan                               = 1 << 0,
    kQTVRValidTilt                              = 1 << 1,
    kQTVRValidFOV                               = 1 << 2,
    kQTVRValidViewCenter                        = 1 << 3
};
```

Constant descriptions

`kQTVRValidPan` If this bit is set, the destination pan angle is used.

`kQTVRValidTilt` If this bit is set, the destination tilt angle is used.

`kQTVRValidFOV` If this bit is set, the destination field of view is used.

`kQTVRValidViewCenter`
If this bit is set, the destination view center is used.

URL Hot Spot Atom

The **URL hot spot atom** has an atom type of `kQTVRHotSpotURLType` (`'url '`). The URL hot spot atom contains a URL string for a particular World Wide Web location (for example, “http://quicktimevr.apple.com”). QuickTime VR automatically links to this URL when the hot spot is clicked. To find out more about exactly what this means see the description of `mcActionLinkToURL` in *QuickTime 3 Reference*.

Example: Getting the Name of a Node

You can use standard QuickTime atom container functions to retrieve the information in a node header atom. For example, the `MyGetNodeName` function defined in Listing 4-1 returns the name of a node, given its node ID.

Listing 4-1 Getting a node’s name

```
OSErr MyGetNodeName (QTVRInstance theInstance, UInt32 theNodeID,
                                                    StringPtr theStringPtr)
{
    OSErr                   theErr = noErr;
    QTAtomContainer         theNodeInfo;
    VRNodeHeaderAtomPtr     theNodeHeader;
```

```
    QTAtom                  theNodeHeaderAtom = 0;

    //Get the node information atom container.
    theErr = QTVRGetNodeInfo(theInstance, theNodeID, &theNodeInfo);

    //Get the node header atom.
    if (!theErr)
        theNodeHeaderAtom = QTFindChildByID(theNodeInfo, kParentAtomIsContainer,
                                                kQTVRNodeHeaderAtomType, 1, nil);
    if (theNodeHeaderAtom != 0) {
        QTLockContainer(theNodeInfo);

        //Get a pointer to the node header atom data.
        theErr = QTGetAtomDataPtr(theNodeInfo, theNodeHeaderAtom, nil,
                                                        (Ptr *)&theNodeHeader);
        //See if there is a name atom.
        if (!theErr && theNodeHeader->nameAtomID != 0) {
            QTAtom theNameAtom;
            theNameAtom = QTFindChildByID(theNodeInfo, kParentAtomIsContainer,
                                kQTVRStringAtomType, theNodeHeader->nameAtomID, nil);
            if (theNameAtom != 0) {
                VRStringAtomPtr theStringAtomPtr;

                //Get a pointer to the name atom data; copy it into the string.
                theErr = QTGetAtomDataPtr(theNodeInfo, theNameAtom, nil,
                                                        (Ptr *)&theStringAtomPtr);
                if (!theErr) {
                    short theLen = theStringAtomPtr->stringLength;
                    if (theLen > 255)
                        theLen = 255;
                    BlockMove(theStringAtomPtr->string, &theStringPtr[1], theLen);
                    theStringPtr[0] = theLen;
                }
            }
        }
        QTUnlockContainer(theNodeInfo);
    }

    QTDisposeAtomContainer(theNodeInfo);
    return(theErr);
}
```

The `MyGetNodeName` function defined in Listing 4-1 retrieves the node information atom container (by calling `QTVRGetNodeInfo`) and then looks inside that container for the node header atom with atom ID 1. If it finds one, it locks the container and then gets a pointer to the node header atom data. The desired information, the node name, is contained in the string atom whose atom ID is specified by the `nameAtomID` field of the node header structure. Accordingly, the `MyGetNodeName` function then calls `QTFindChildByID` once again to find that string atom. If the string atom is found, `MyGetNodeName` calls `QTGetAtomDataPtr` to get a pointer to the string atom data. Finally, `MyGetNodeName` copies the string data into the appropriate location and cleans up after itself before returning.

Custom Atoms

The author of a QuickTime VR movie may choose to add custom atoms to either the VR world or node information atom containers. Those atoms can be extracted within an application to provide additional information that the application may use.

Information that pertains to the entire scene might be stored in a custom atom within the VR world atom container. Node-specific information could be stored in the individual node information atom containers or as sibling atoms to the node location atoms within the VR world.

Custom hot spot atoms should be stored as siblings to the hot spot information atoms in the node information atom container. Generally, its atom type is the same as the custom hot spot type. You can set up an intercept procedure in your application in order to process clicks on the custom hot spots.

If you use custom atoms, you should install your hot spot intercept procedure when you open the movie. Listing 4-2 is an example of such an intercept procedure.

Listing 4-2 Typical hot spot intercept procedure

```
QTVRInterceptProc MyProc = NewQTVRInterceptProc (MyHotSpot);
QTVRInstallInterceptProc (qtvr, kQTVRTriggerHotSpotSelector, myProc, 0, 0);

pascal void MyHotSpot (QTVRInstance qtvr, QTVRInterceptPtr qtvrMsg,
                       SInt32 refCon, Boolean *cancel)
{
```

```
    UInt32 hotSpotID = (UInt32) qtvrMsg->parameter[0];
    QTAtomContainer nodeInfo =
            (QTAtomContainer) qtvrMsg->parameter[1];
    QTAtom hotSpotAtom = (QTAtom) qtvrMsg->parameter[2];
    OSType hotSpotType;
    CustomData myCustomData;
    QTAtom myAtom;

    QTVRGetHotSpotType (qtvr, hotSpotID, &hotSpotType);
    if (hotSpotType != kMyAtomType) return;

    // It's our type of hot spot - don't let anyone else handle it
    *cancel = true;

    // Find our custom atom
    myAtom = QTFindChildByID (nodeInfo, hotSpotAtom, kMyAtomType, 1, nil);
    if (myAtom != 0) {
        OSErr err;
        // Copy the custom data into our structure
        err = QTCopyAtomDataToPtr (nodeInfo, myAtom, false,
                        sizeof(CustomData), &myCustomData, nil);
        if (err == noErr)
            // Do something with it
            DoMyHotSpotStuff (hotSpotID, &myCustomData);
    }
}
```

Your intercept procedure is called for clicks on any hot spot. You should check to see if it is your type of hot spot and if so, extract the custom hot spot atom and do whatever is appropriate for your hot spot type (`DoMyHotSpotStuff`).

When you no longer need the intercept procedure you should call `QTVRInstallInterceptProc` again with the same selector and a `nil` procedure pointer and then call `DisposeRoutineDescriptor` on `myProc`.

Note
Apple reserves all hot spot and atom types with lowercase letters. Your custom hot spot type should contain all uppercase letters. ◆

CHAPTER 5

Creating QuickTime VR Movies

Contents

This chapter describes the format of the tracks that make up a QuickTime VR movie file. The information in this chapter, combined with the information in Chapter 4, "QuickTime VR Atom Containers," and the overview from Chapter 1, "About QuickTime VR," will enable you to add to your application the ability to create QuickTime VR movies. This chapter uses the term QuickTime VR file format, even though this is a bit of a misnomer, because the chapter does not describe the details of exactly where data is stored in the file nor does it describe the format of the standard QuickTime movie atoms. That information is available in *Inside Macintosh: QuickTime*. However, you do not need to know that level of detail to create QuickTime VR movies, because QuickTime provides several routines for creating QuickTime movies. You do need to be familiar with the sections of that book that describe how to create movies and how to work with media samples. You should also be familiar with track references as described in *QuickTime 3 Reference* .

IMPORTANT

This chapter describes the file format supported by version 2.1 of the QuickTime VR Manager. For information on the file format supported by earlier versions of QuickTime VR, see Macintosh Technotes numbers 1035 and 1036. The Macintosh Technotes are available electronically on the *Developer CD Series* and on the Technote Web site at http://devworld.apple.com/dev/technotes.shtml ▲

Components of a QuickTime VR Movie

A QuickTime VR movie is stored on disk in a format known as the **QuickTime VR file format.** Beginning in QuickTime VR 2.0, a QuickTime VR movie can contain one or more nodes. Each node is either a panorama or an object. In addition, a QuickTime VR movie can contain various types of hot spots including links between any two types of nodes.

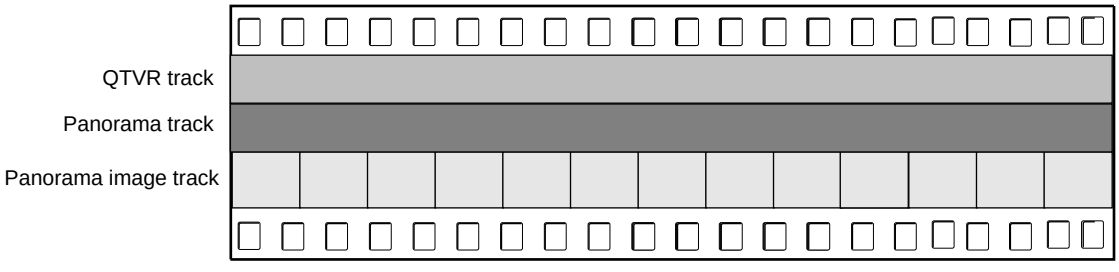
All QuickTime VR movies contain a single **QTVR track,** a special type of QuickTime track that maintains a list of the nodes in the movie. Each individual sample in a QTVR track contains general information and hot spot information for a particular node. If a QuickTime VR movie contains any panoramic nodes, that movie also contains a single **panorama track,** and if it contains any object nodes, it also contains a single **object track.** The panorama and object tracks

contain information specific to the panoramas or objects in the movie. The actual image data for both panoramas and objects is usually stored in standard QuickTime video tracks, hereafter referred to as **image tracks.** (An image track can also be any type of track that is capable of displaying an image, such as a QuickTime 3D track.) The individual frames in the image track for a panorama make up the diced frames of the original single panoramic image. The frames for the image track of an object represent the many different views of the object. Hot spot image data is stored in parallel video tracks for both panoramas and objects.

Single-Node Panoramic Movies

Figure 5-1 illustrates the basic structure of a single-node panoramic movie. As you can see, every panoramic movie contains at least three tracks: (1) a QTVR track, (2) a panorama track, and (3) a panorama image track.

Figure 5-1 The structure of a single-node panoramic movie file



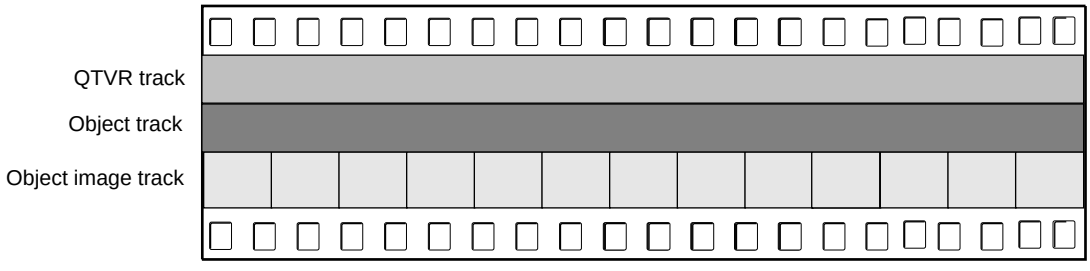
For a single-node panoramic movie, the QTVR track contains just one sample. There is a corresponding sample in the panorama track, whose time and duration are the same as the time and duration of the sample in the QTVR track. The time base of the movie is used to locate the proper video samples in the panorama image track. For a panoramic movie, the video sample for the first diced frame of a node's panoramic image is located at the same time as the corresponding QTVR and panorama track samples. The total duration of all the video samples is the same as the duration of the corresponding QTVR sample and the panorama sample.

A panoramic movie can contain an optional hot spot image track and any number of standard QuickTime tracks. A panoramic movie can also contain panoramic image tracks with a lower resolution. The video samples in these low-resolution image tracks must be located at the same time and must have the same total duration as the QTVR track. Likewise, the video samples for a hot spot image track, if one exists, must be located at the same time and must have the same total duration as the QTVR track.

Single-Node Object Movies

Figure 5-2 illustrates the basic structure of a single-node object movie. As you can see, every object movie contains at least three tracks: (1) a QTVR track, (2) an object track, and (3) an object image track.

Figure 5-2 The structure of a single-node object movie file



For a single-node object movie, the QTVR track contains just one sample. There is a corresponding sample in the object track, whose time and duration are the same as the time and duration of the sample in the QTVR track. The time base of the movie is used to locate the proper video samples in the object image track.

For an object movie, the frame corresponding to the first row and column in the object image array is located at the same time as the corresponding QTVR and object track samples. The total duration of all the video samples is the same as the duration of the corresponding QTVR sample and the object sample.

In addition to these three required tracks, an object movie can also contain a hot spot image track and any number of standard QuickTime tracks (such as video, sound, and text tracks). A hot spot image track for an object is a QuickTime

video track that contains images of colored regions delineating the hot spots; an image in the hot spot image track must be synchronized to match the appropriate image in the object image track. A hot spot image track should be 8 bits deep and can be compressed with any lossless compressor (including temporal compressors).

Note
To assign a single fixed-position hot spot to all views of an object, you should create a hot spot image track that consists of a single video frame whose duration is the entire node time. ◆

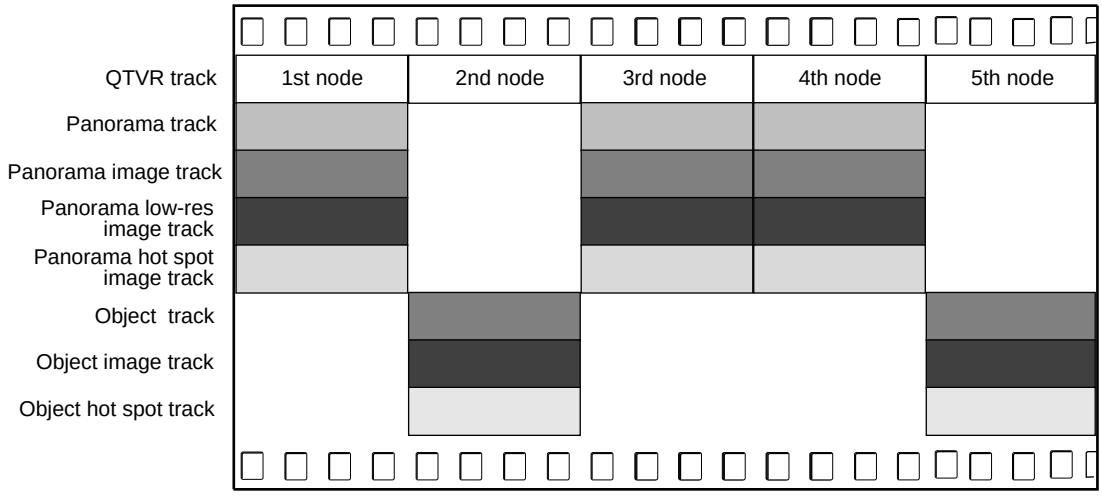
To play a time-based track with the object movie, you must synchronize the sample data of that track to the start and stop times of a view in the object image track. For example, to play a different sound with each view of an object, you might store a sound track in the movie file with each set of sound samples synchronized to play at the same time as the corresponding object's view image. (This technique also works for video samples.) Another way to add sound or video is simply to play a sound or video track during the object's view animation; to do this, you need to add an active track to the object that is equal in duration to the object's row duration.

IMPORTANT
In a QuickTime VR movie file, the panorama image tracks and panorama hot spot tracks must be disabled. For an object, the object image tracks must be enabled and the object hot spot tracks must be disabled. ▲

Multinode Movies

A multinode QuickTime VR movie can contain any number of object and panoramic nodes. Figure 5-3 illustrates the structure of a QuickTime VR movie that contains five nodes (in this case, three panoramic nodes and two object nodes).

Figure 5-3 The structure of a multinode movie file



IMPORTANT

Panoramic tracks and object tracks must never be located at the same time. ▲

QuickTime VR Movie Creation

A QuickTime VR movie file is a QuickTime movie file. The only differences between a QuickTime VR movie file and a typical time-based QuickTime movie file are the type and usage of the tracks contained in the movie and the necessary QuickTime user data attached to the movie. In particular, for the Mac OS, the file type should be `'MooV'`; for multiple platform compatibility, the file extension should be `.mov`.

When you create a QuickTime VR movie, you need to add a special piece of user data that identifies which movie controller to invoke for this movie. The movie controller type for QuickTime VR movies is `'qtvr'`. This user data is examined by the Movie Toolbox when an application calls the `NewMovieController` function for that movie. Listing 5-1 shows how to add the appropriate user data to a new movie.

Listing 5-1 Specifying the QuickTime VR movie controller

```
UserData            myUserData;
OSType              controllerSubType = FOUR_CHAR_CODE('qtvr');

myUserData = GetMovieUserData(theMovie);
SetUserDataItem(myUserData, &controllerSubType,
    sizeof(controllerSubType), kQTControllerType, kQTControllerID);
```

The constants `kQTControllerType` and `kQTControllerID` are defined by QuickTime VR:

```
enum {
    kQTControllerType                           = FOUR_CHAR_CODE('ctyp').
    kQTControllerID                             = 1
};
```

Also, as with any QuickTime movie that is intended to be played on multiple operating systems, a data fork version of the file should be created using the `FlattenMovie` function with the `flattenAddMovieToDataFork` flag set. Note that the resulting file is optimized for random access and might not perform well in an environment that requires streaming access (such as Web browsing). The section "Optimizing QuickTime VR Movies for Web Playback" (page 286) describes an export component provided by Apple that allows you to optimize a QuickTime VR file for the Web and includes the ability to store a small preview of the movie in the file.

Track References

QuickTime VR uses track references to establish a relationship between the various tracks in the file. The `AddTrackReference` routine is used to create the reference. A reference to a particular track is identified by its reference type and index. When you add a track reference, you supply the type and QuickTime returns an index, which always starts at 1 for a particular type. The QTVR track contains a reference to the panorama and object tracks in the movie. The reference types are `kQTVRPanoramaType` (`'pano'`) and `kQTVRObjectType` (`'obje'`). To add a reference to the panorama track from the QTVR track you would use the following line of code:

```
err = AddTrackReference (qtvrTrack, panoTrack, kQTVRPanoramaType,
&index);
```

Because there are at most one panorama track and one object track in QuickTime VR 2.1 movies, the index is always one and hence is not stored in the file. The panorama and object tracks in turn contain track references to their image and hot spot tracks. The reference types for these are `kQTVRImageTrackRefType` (`'imgt'`) and `kQTVRHotSpotTrackRefType` (`'hott'`). There can be many image and hot spot tracks, so the track reference indexes returned by the `AddTrackReference` call are stored in data structures in the panorama and object tracks as described later in this chapter.

The QTVR Track

As mentioned in “Components of a QuickTime VR Movie” (page 263), the QTVR track is a special type of QuickTime track that maintains a list of all the nodes in a movie. The media type for a QTVR track is `'qtvr'`. All the media samples in a QTVR track share a common sample description. This sample description contains the VR world atom container. The track contains one media sample for each node in the movie. Each QuickTime VR media sample contains a node information atom container, also described in Chapter 4, “QuickTime VR Atom Containers.”

The QuickTime VR Sample Description Structure

Whereas the QuickTime VR media sample is simply the node information itself, all sample descriptions are required by QuickTime to have a certain structure for the first several bytes. The structure for the QuickTime VR sample description is as follows.

```
typedef struct QTVRSampleDescription {
    UInt32                              size;
    UInt32                              type;
    UInt32                              reserved1;
    UInt16                              reserved2;
    UInt16                              dataRefIndex;
```

```
    UInt32                              data;
} QTVRSampleDescription, *QTVRSampleDescriptionPtr,
**QTVRSampleDescriptionHandle;
```

Field descriptions

`size`	The size, in bytes, of the sample description header structure, including the VR world atom container contained in the `data` field.
`type`	The sample description type. For QuickTime VR movies, this type should be `'qtvr'`.
`reserved1`	Reserved. This field must be 0.
`reserved2`	Reserved. This field must be 0.
`dataRefIndex`	Reserved. This field must be 0.
`data`	The VR world atom container. The sample description structure is extended to hold this atom container.

Example: Adding Atom Containers

Assuming you have already created the VR world and node information atom containers, Listing 5-2 shows the code (minus error checking) you would use to add them to the QTVR track.

Listing 5-2 Adding atom containers to a track

```
long descSize;
QTVRSampleDescriptionHandle qtvrSampleDesc;

// Create a QTVR sample description handle
descSize = sizeof(QTVRSampleDescription) + GetHandleSize((Handle) vrWorld) -
                                    sizeof(UInt32);
qtvrSampleDesc = (QTVRSampleDescriptionHandle) NewHandleClear (descSize);
(*qtvrSampleDesc)->size = descSize;
(*qtvrSampleDesc)->type = kQTVRQTVRType;

// Copy the vrWorld atom container data into the QTVR sample description
BlockMove (*((Handle) vrWorld), &((*qtvrSampleDesc)->data),
                          GetHandleSize((Handle) vrWorld));
// Now add it to the QTVR track's media
```

```
err = BeginMediaEdits (qtvrMedia);
err = AddMediaSample (qtvrMedia, (Handle) nodeInfo, 0,
    GetHandleSize((Handle) nodeInfo), duration,
    (SampleDescriptionHandle) qtvrSampleDesc, 1, 0, &sampleTime);
err = EndMediaEdits (qtvrMedia);
InsertMediaIntoTrack (qtvrTrack, trackTime, sampleTime, duration, 1L<<16);
```

The `duration` value is computed based on the duration of the corresponding image track samples for the node. The value of `trackTime` is the time for the beginning of the current node (zero for a single node movie). The values of `duration` and `sampleTime` are in the time base of the media; the value of `trackTime` is in the movie's time base.

Panorama Tracks

A movie's **panorama track** is a track that contains information about the panoramic nodes in a scene. The media type of the panorama track is `'pano'`. Each sample in a panorama track corresponds to a single panoramic node. This sample parallels the corresponding sample in the QTVR track. Panorama tracks do not have a sample description (although QuickTime requires that you specify a dummy sample description when you call `AddMediaSample` to add a sample to a panorama track). The sample itself contains an atom container that includes a panorama sample atom and other optional atoms.

Panorama Sample Atom Structure

A panorama sample atom has an atom type of `kQTVRPanoSampleDataAtomType` (`'pdat'`). It describes a single panorama, including track reference indexes of the scene and hot spot tracks and information about the default viewing angles and the source panoramic image.

The structure of a panorama sample atom is defined by the `VRPanoSampleAtom` data type:

```
typedef struct VRPanoSampleAtom {
    UInt16                          majorVersion;
    UInt16                          minorVersion;
    UInt32                          imageRefTrackIndex;
    UInt32                          hotSpotRefTrackIndex;
    Float32                         minPan;
    Float32                         maxPan;
```

```
    Float32                             minTilt;
    Float32                             maxTilt;
    Float32                             minFieldOfView;
    Float32                             maxFieldOfView;
    Float32                             defaultPan;
    Float32                             defaultTilt;
    Float32                             defaultFieldOfView;
    UInt32                              imageSizeX;
    UInt32                              imageSizeY;
    UInt16                              imageNumFramesX;
    UInt16                              imageNumFramesY;
    UInt32                              hotSpotSizeX;
    UInt32                              hotSpotSizeY;
    UInt16                              hotSpotNumFramesX;
    UInt16                              hotSpotNumFramesY;
    UInt32                              flags;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRPanoSampleAtom, *VRPanoSampleAtomPtr;
```

Field descriptions

`majorVersion` The major version number of the file format.

`minorVersion` The minor version number of the file format.

`imageRefTrackIndex` The index of the image track reference. This is the index returned by the `AddTrackReference` function when the image track is added as a reference to the panorama track. There can be more than one image track for a given panorama track and hence multiple references. (A panorama track might have multiple image tracks if the panoramas have different characteristics, which could occur if the panoramas were shot with different size camera lenses.) The value in this field is 0 if there is no corresponding image track.

`hotSpotRefTrackIndex` The index of the hot spot track reference.

`minPan` The minimum pan angle, in degrees. For a full panorama, the value of this field is usually 0.0.

`maxPan` The maximum pan angle, in degrees. For a full panorama, the value of this field is usually 360.0.

`minTilt`	The minimum tilt angle, in degrees. For a high-resolution panorama, a typical value for this field is –42.5.
`maxTilt`	The maximum tilt angle, in degrees. For a high-resolution panorama, a typical value for this field is +42.5.
`minFieldOfView`	The minimum vertical field of view, in degrees. For a high-resolution panorama, a typical value for this field is 5.0. The value in this field is 0 for the default minimum field of view, which is 5 percent of the maximum field of view.
`maxFieldOfView`	The maximum vertical field of view, in degrees. For a high-resolution panorama, a typical value for this field is 85.0. The value in this field is 0 for the default maximum field of view, which is `maxTilt – minTilt`.
`defaultPan`	The default pan angle, in degrees.
`defaultTilt`	The default tilt angle, in degrees.
`defaultFieldOfView`	The default vertical field of view, in degrees.
`imageSizeX`	The width, in pixels, of the panorama stored in the highest resolution image track. For a high-resolution panorama, a typical value for this field is 768.
`imageSizeY`	The height, in pixels, of the panorama stored in the highest resolution image track. For a high-resolution panorama, a typical value for this field is 2496.
`imageNumFramesX`	The number of frames into which the panoramic image is diced horizontally. The width of each frame (which is `imageSizeX / imageNumFramesX`) should be divisible by 4. For a high-resolution panorama, a typical value for this field is 1.
`imageNumFramesY`	The number of frames into which the panoramic image is diced vertically. The height of each frame (which is `imageSizeY / imageNumFramesY`) should be divisible by 4. For a high-resolution panorama, a typical value for this field is 24.
`hotSpotSizeX`	The width, in pixels, of the panorama stored in the highest resolution hot spot image track.
`hotSpotSizeY`	The height, in pixels, of the panorama stored in the highest resolution hot spot image track.

`hotSpotNumFramesX`	The number of frames into which the panoramic image is diced horizontally for the hot spot image track. For a high-resolution panorama, a typical value for this field is 1.
`hotSpotNumFramesY`	The number of frames into which the panoramic image is diced vertically for the hot spot image track. For a high-resolution panorama, a typical value for this field is 24.
`flags`	A set of panorama flags. The only currently defined flag is `kQTVRPanoFlagHorizontal`, which indicates that the panorama is oriented horizontally. This orientation is not supported in QuickTime VR 2.1.
`reserved1`	Reserved. This field must be 0.
`reserved2`	Reserved. This field must be 0.

The minimum and maximum values in the panorama sample atom describe the physical limits of the panoramic image. QuickTime VR allows you to set further constraints on what portion of the image a user will see by calling the `QTVRSetConstraints` routine. You can also preset image constraints by adding constraint atoms to the panorama sample atom container. The three constraint atom types are `kQTVRPanConstraintAtomType`, `kQTVRTiltConstraintAtomType`, and `kQTVRFOVConstraintAtomType`. Each of these atom types share a common structure defined by the `VRAngleRangeAtom` data type:

```
typedef struct VRAngleRangeAtom {
    Float32                             minimumAngle;
    Float32                             maximumAngle;
} VRAngleRangeAtom, *VRAngleRangeAtomPtr;
```

Field descriptions

`minimumAngle`	The minimum angle in the range, in degrees.
`maximumAngle`	The maximum angle in the range, in degrees.

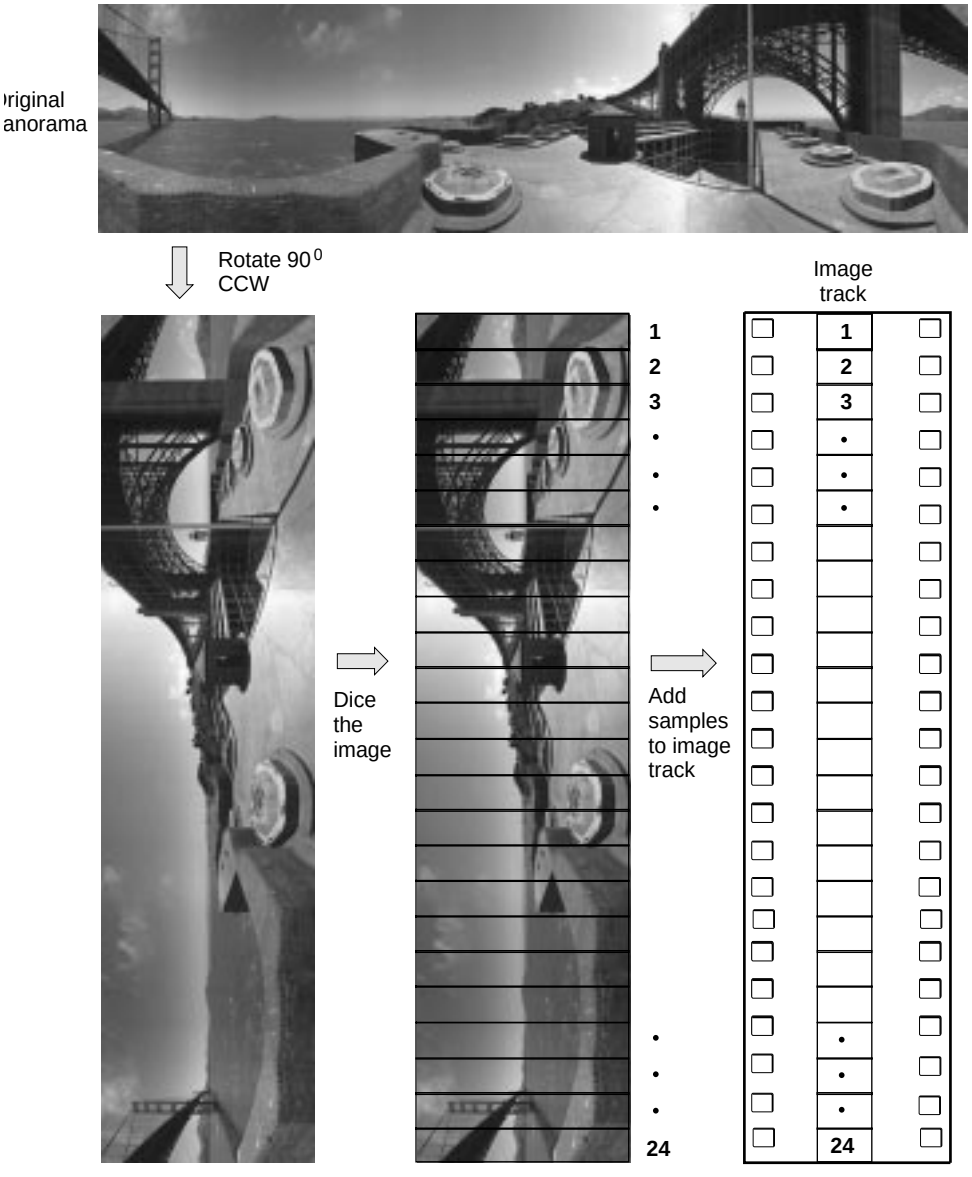
The actual panoramic image for a panoramic node is contained in a **panorama image track,** which is a standard QuickTime video track. The track reference to this track is stored in the `imageRefTrackIndex` field of the panorama sample atom. The panoramic image can be created in many ways. You can use the panorama stitcher provided by the QuickTime VR Authoring Studio to stitch together several photographs. Alternatively, you can use a graphics rendering application or a panoramic camera.

QuickTime VR 2.1 requires the original panoramic image to be rotated 90 degrees counterclockwise. The rotated image must then be diced into smaller frames. Each diced frame is compressed and added to the video track as a video

sample. (See Figure 5-4.) Frames can be compressed using any spatial compressor; temporal compression is not allowed for panoramic image tracks.

Note
A panorama sample atom (which contains information about a single panorama) contains a flag that indicates whether the diced panoramic image is oriented horizontally or vertically. Currently, only vertical orientation is supported. Future versions of QuickTime VR might support horizontal panoramic images. ◆

Figure 5-4 Creating an image track for a panorama



When a panorama contains hot spots, the movie file contains a **hot spot image track,** a video track that contains a parallel panorama with the hot spots designated by colored regions. Each diced frame of the hot spot panoramic image must be compressed with a lossless compressor (such as QuickTime's graphics compressor). The dimensions of the hot spot panoramic image are usually the same as those of the image track's panoramic image, but this is not required. The dimensions must, however, have the same aspect ratio as the image track's panoramic image. A hot spot image track should be 8 bits deep.

It's possible to store one or more low-resolution versions of a panoramic image in a movie file; those versions are called **low-resolution image tracks.** If there is not enough memory at runtime to use the normal image track, QuickTime VR uses a lower resolution image track if one is available. A low-resolution image track contains diced frames just like the higher resolution track, but the reconstructed panoramic image is half the height and half the width of the higher resolution image. The number of diced frames in the lower resolution image track is usually half that in the normal image track.

IMPORTANT

In QuickTime VR 2.1, the panoramic images in the lower resolution image tracks and the hot spot image tracks, if present, must be rotated 90 degrees counterclockwise (just like images in the panorama image track). ▲

Track Reference Entry Structure

Since there are no fields in the sample track atom to indicate the presence of low-resolution image tracks, a separate sibling atom must be added to the panorama sample atom container. The track reference array atom contains an array of track reference entry structures that specify information about any low-resolution image tracks contained in a movie. Its atom type is `kQTVRTrackRefArrayAtomType` (`'tref'`).

A track reference entry structure is defined by the `VRTrackRefEntry` data type:

```
typedef struct VRTrackRefEntry {
    UInt32                              trackRefType;
    UInt16                              trackResolution;
    UInt32                              trackRefIndex;
} VRTrackRefEntry;
```

Field descriptions

`trackRefType`	The track reference type. See "Track References" (page 268) for a description of the available track reference types.
`trackResolution`	The track resolution. See "Resolutions" (page 83) for a description of the values that can occur in this field.
`trackRefIndex`	The index of the track reference.

The number of entries in the track reference array atom is determined by dividing the size of the atom by `sizeof` (`VRTrackRefEntry`).

`kQTVRPreviewTrackRes` is a special value for the `trackResolution` field in the `VRTrackRefEntry` structure. This is used to indicate the presence of a special preview image track. Preview image tracks are discussed in "Optimizing QuickTime VR Movies for Web Playback" (page 286).

Object Tracks

A movie's **object track** is a track that contains information about the object nodes in a scene. The media type of the object track is `'obje'`. Each sample in an object track corresponds to a single object node in the scene. The samples of the object track contain information describing the object images stored in the object image track. These object information samples parallel the corresponding node samples in the QTVR track and are equal in time and duration to a particular object node's image samples in the object's image track as well as the object node's hot spot samples in the object's hot spot track.

Object tracks do not have a sample description (although QuickTime requires that you specify a dummy sample description when you call `AddMediaSample` to add a sample to an object track). The sample itself is an atom container that contains a single object sample atom and other optional atoms.

Object Sample Atom Structure

An object sample atom describes a single object, including information about the default viewing angles and the view settings. The structure of an object sample atom is defined by the `VRObjectSampleAtom` data type:

```
typedef struct VRObjectSampleAtom {
    UInt16                          majorVersion;
    UInt16                          minorVersion;
    UInt16                          movieType;
    UInt16                          viewStateCount;
```

```
    UInt16                              defaultViewState;
    UInt16                              mouseDownViewState;
    UInt32                              viewDuration;
    UInt32                              columns;
    UInt32                              rows;
    Float32                             mouseMotionScale;
    Float32                             minPan;
    Float32                             maxPan;
    Float32                             defaultPan;
    Float32                             minTilt;
    Float32                             maxTilt;
    Float32                             defaultTilt;
    Float32                             minFieldOfView;
    Float32                             fieldOfView;
    Float32                             defaultFieldOfView;
    Float32                             defaultViewCenterH;
    Float32                             defaultViewCenterV;
    Float32                             viewRate;
    Float32                             frameRate;
    UInt32                              animationSettings;
    UInt32                              controlSettings;
} VRObjectSampleAtom, *VRObjectSampleAtomPtr;
```

Field descriptions

`majorVersion` The major version number of the file format.

`minorVersion` The minor version number of the file format.

`movieType` The movie controller type. See “Object Controller Types” (page 281) for a description of the available movie controllers for object views.

`viewStateCount` The number of view states of the object. A view state selects an alternate set of images for an object’s views. The value of this field must be positive.

`defaultViewState` The 1-based index of the default view state. The default view state image for a given view is displayed when the mouse button is not down.

`mouseDownViewState` The 1-based index of the mouse-down view state. The mouse-down view state image for a given view is displayed while the user holds the mouse button down while the cursor is over an object movie.

`viewDuration`	The total movie duration of all image frames contained in an object's view. In an object that uses a single frame to represent a view, the duration is the image track's sample duration time.
`columns`	The number of columns in the object image array (that is, the number of horizontal positions or increments in the range defined by the minimum and maximum pan values). The value of this field must be positive.
`rows`	The number of rows in the object image array (that is, the number of vertical positions or increments in the range defined by the minimum and maximum tilt values). The value of this field must be positive.
`mouseMotionScale`	The mouse motion scale factor (that is, the number of degrees that an object is panned or tilted when the cursor is dragged the entire width of the VR movie image). The default value is 180.0.
`minPan`	The minimum pan angle, in degrees. The value of this field must be less than the value of the `maxPan` field.
`maxPan`	The maximum pan angle, in degrees. The value of this field must be greater than the value of the `minPan` field.
`defaultPan`	The default pan angle, in degrees. This is the pan angle used when the object is first displayed. The value of this field must be greater than or equal to the value of the `minPan` field and less than or equal to the value of the `maxPan` field.
`minTilt`	The minimum tilt angle, in degrees. The default value is +90.0. The value of this field must be less than the value of the `maxTilt` field.
`maxTilt`	The maximum tilt angle, in degrees. The default value is –90.0. The value of this field must be greater than the value of the `minTilt` field.
`defaultTilt`	The default tilt angle, in degrees. This is the tilt angle used when the object is first displayed. The value of this field must be greater than or equal to the value of the `minTilt` field and less than or equal to the value of the `maxTilt` field.
`minFieldOfView`	The minimum field of view to which the object can zoom. The valid range for this field is from 1 to the value of the `fieldOfView` field. The value of this field must be positive.

`fieldOfView`	The image field of view, in degrees, for the entire object. The value in this field must be greater than or equal to the value of the `minFieldOfView` field.
`defaultFieldOfView`	The default field of view for the object. This is the field of view used when the object is first displayed. The value in this field must be greater than or equal to the value of the `minFieldOfView` field and less than or equal to the value of the `fieldOfView` field.
`defaultViewCenterH`	The default horizontal view center.
`defaultViewCenterV`	The default vertical view center.
`viewRate`	The view rate (that is, the positive or negative rate at which the view animation in objects plays, if view animation is enabled). The value of this field must be from –100.0 through +100.0, inclusive.
`frameRate`	The frame rate (that is, the positive or negative rate at which the frame animation in a view plays, if frame animation is enabled). The value of this field must be from –100.0 through +100.0, inclusive.
`animationSettings`	A set of 32-bit flags that encode information about the animation settings of the object. See “Animation Settings” (page 282) for a description of the constants you can use to specify a value for this field.
`controlSettings`	A set of 32-bit flags that encode information about the control settings of the object. See “Control Settings” (page 283) for a description of the constants you can use to specify a value for this field.

Object Controller Types

The `movieType` field of the object sample atom structure specifies an object controller type, that is, the user interface to be used to manipulate the object.

QuickTime VR supports the following controller types:

```
enum ObjectUITypes {
    kGrabberScrollerUI                  = 1,
    kOldJoyStickUI                      = 2,
    kJoystickUI                         = 3,
```

```
    kGrabberUI                                  = 4,
    kAbsoluteUI                                 = 5
};
```

Constant descriptions

`kGrabberScrollerUI` The default controller, which displays a hand for dragging and rotation arrows when the cursor is along the edges of the object window.

`kOldJoyStickUI` A joystick controller, which displays a joystick-like interface for spinning the object. With this controller, the direction of panning is reversed from the direction of the grabber.

`kJoystickUI` A joystick controller, which displays a joystick-like interface for spinning the object. With this controller, the direction of panning is consistent with the direction of the grabber.

`kGrabberUI` A grabber-only interface, which displays a hand for dragging but does *not* display rotation arrows when the cursor is along the edges of the object window.

`kAbsoluteUI` An absolute controller, which displays a finger for pointing. The absolute controller switches views based on a row-and-column grid mapped into the object window.

Animation Settings

The `animationSettings` field of the object sample atom is a long integer that specifies a set of animation settings for an object node. Animation settings specify characteristics of the movie while it is playing. You can use these constants to specify animation settings:

```
enum QTVRAnimationSettings {
    kQTVRObjectAnimateViewFramesOn              = (1 << 0),
    kQTVRObjectPalindromeViewFramesOn           = (1 << 1),
    kQTVRObjectStartFirstViewFrameOn            = (1 << 2),
    kQTVRObjectAnimateViewsOn                   = (1 << 3),
    kQTVRObjectPalindromeViewsOn                = (1 << 4),
    kQTVRObjectSyncViewToFrameRate              = (1 << 5),
    kQTVRObjectDontLoopViewFramesOn             = (1 << 6),
    kQTVRObjectPlayEveryViewFrameOn             = (1 << 7)
};
```

Constant descriptions

`kQTVRObjectAnimateViewFramesOn`
: If this bit is set, play all frames in the current view state.

`kQTVRObjectPalindromeViewFramesOn`
: If this bit is set, play a back-and-forth animation of the frames of the current view state.

`kQTVRObjectStartFirstViewFrameOn`
: If this bit is set, play the frame animation starting with the first frame in the view (that is, at the view start time).

`kQTVRObjectAnimateViewsOn`
: If this bit is set, play all views of the current object in the default row of views.

`kQTVRObjectPalindromeViewsOn`
: If this bit is set, play a back-and-forth animation of all views of the current object in the default row of views.

`kQTVRObjectSyncViewToFrameRate`
: If this bit is set, synchronize the view animation to the frame animation and use the same options as for frame animation.

`kQTVRObjectDontLoopViewFramesOn`
: If this bit is set, stop playing the frame animation in the current view at the end.

`kQTVRObjectPlayEveryViewFrameOn`
: If this bit is set, play every view frame regardless of play rate. The play rate is used to adjust the duration in which a frame appears but no frames are skipped so the rate is not exact.

Control Settings

The `controlSettings` field of the object sample atom is a long integer that specifies a set of control settings for an object node. Control settings specify whether the object can wrap during panning and tilting, as well as other features of the node. The control settings are specified using these bit flags:

```
enum QTVRControlSettings {
    kQTVRObjectWrapPanOn                    = (1 << 0),
    kQTVRObjectWrapTiltOn                   = (1 << 1),
    kQTVRObjectCanZoomOn                    = (1 << 2),
    kQTVRObjectReverseHControlOn            = (1 << 3),
```

```
    kQTVRObjectReverseVControlOn                = (1 << 4),
    kQTVRObjectSwapHVControlOn                  = (1 << 5),
    kQTVRObjectTranslationOn                    = (1 << 6)
};
```

IMPORTANT

See the section "Control Settings" (page 74), for a more complete description of these control settings. ▲

Constant descriptions

`kQTVRObjectWrapPanOn`
: If this bit is set, enable wrapping during panning. When this control setting is enabled, the user can wrap around from the current pan constraint maximum value to the pan constraint minimum value (or vice versa) using the mouse or arrow keys.

`kQTVRObjectWrapTiltOn`
: If this bit is set, enable wrapping during tilting. When this control setting is enabled, the user can wrap around from the current tilt constraint maximum value to the tilt constraint minimum value (or vice versa) using the mouse or arrow keys.

`kQTVRObjectCanZoomOn`
: If this bit is set, enable zooming. When this control setting is enabled, the user can change the current field of view using the zoom-in and zoom-out keys on the keyboard (or using the VR controller buttons).

`kQTVRObjectReverseHControlOn`
: If this bit is set, reverse the direction of the horizontal control.

`kQTVRObjectReverseVControlOn`
: If this bit is set, reverse the direction of the vertical control.

`kQTVRObjectSwapHVControlOn`
: If this bit is set, exchange the horizontal and vertical controls.

`kQTVRObjectTranslationOn`
: If this bit is set, enable translation. When this setting is enabled, the user can translate using the mouse when

either the translate key is held down or the controller translation mode button is toggled on.

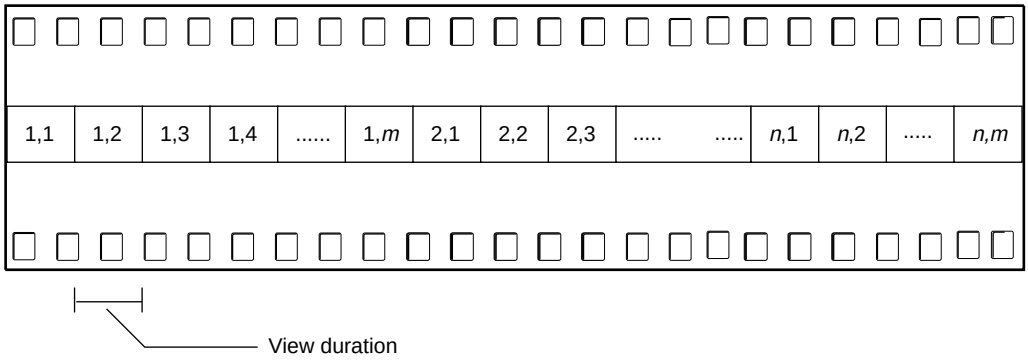
Track References for Object Tracks

The track references to an object's image and hot spot tracks are not handled the same way as track references to panoramas. The track reference types are the same (`kQTVRImageTrackRefType` and `kQTVRHotSpotTrackRefAtomType`), but the location of the reference indexes is different. There is no entry in the object sample atom for the track reference indexes. Instead, separate atoms using the `VRTrackRefEntry` structure are stored as siblings to the object sample atom. The types of these atoms are `kQTVRImageTrackRefAtomType` and `kQTVRHotSpotTrackRefAtomType`. If either of these atoms is not present, then the reference index to the corresponding track is assumed to be one.

Note
The `trackResolution` field in the `VRTrackRefEntry` structure is currently ignored for object tracks. ◆

The actual views of an object for an object node are contained in an **object image track,** which is usually a standard QuickTime video track. (An object image track can also be any type of track that is capable of displaying an image, such as a QuickTime 3D track.) As described in the section "Object Nodes" (page 24) in Chapter 1, "About QuickTime VR,"these views are often captured by moving a camera around the object in a defined pattern of pan and tilt angles. The views must then be ordered into an object image array, which is stored as a one-dimensional sequence of frames in the movie's video track (see Figure 5-5).

Figure 5-5 The structure of an image track for an object



For object movies containing frame animation, each animated view in the object image array consists of the animating frames. It is not necessary that each view in the object image array contain the same number of frames, but the view duration of all views in the object movie must be the same.

For object movies containing alternate view states, alternate view states are stored as separate object image arrays that immediately follow the preceding view state in the object image track. Each state does not need to contain the same number of frames. However, the total movie time of each view state in an object node must be the same.

Optimizing QuickTime VR Movies for Web Playback

All of the discussion so far is sufficient to allow you to create a valid QuickTime VR movie. However, there are special considerations when the movie is to be played back on the Internet. Originally, both QuickTime movies and QuickTime VR movies had to be completely downloaded to the user's local hard disk before they could be viewed. Starting with QuickTime 2.5, if the movie data is properly laid out in the file, standard linear QuickTime movies can be viewed almost immediately. The frames that have been downloaded so far are shown while subsequent frames continue to be downloaded.

The important change that took place to allow this to happen was for QuickTime to place global movie information at the beginning of the file.

Originally it was at the end of the file. After that, the frame data simply needs to be in order in the file. Similarly, QuickTime VR files also need to be laid out in a certain manner in order to get some sort of quick feedback when viewing on the Web. Roughly speaking this involves writing out all of the media samples in the file in a particular order. Apple now provides a movie export component that does this for you: the QTVR Flattener.

The QTVR Flattener

The QTVR Flattener is a movie export component that converts an existing QuickTime VR single node movie into a new movie that is optimized for the Web. Not only does the flattener reorder the media samples, but for panoramas it also creates a small preview of the panorama. When viewed on the Web, this preview appears after 5% to 10% of the movie data has been downloaded, allowing users to see a lower resolution version of the panorama.

Using the QTVR Flattener from your application is quite easy. After you have created the QuickTime VR movie, you simply open the QTVR Flattener component and call the `MovieExportToFile` routine as shown in Listing 5-3.

Listing 5-3 Using the flattener

```
ComponentDescription desc;
Component flattener;
ComponentInstance qtvrExport = nil;

desc.componentType = MovieExportType;
desc.componentSubType = MovieFileType;
desc.componentManufacturer = QTVRFlattenerType;

flattener = FindNextComponent(nil, &desc);
if (flattener) qtvrExport = OpenComponent (flattener);

if (qtvrExport)
    MovieExportToFile (qtvrExport, &myFileSpec, myQTVRMovie, nil, 0, 0);
```

The code fragment shown in Listing 5-3 creates a flattened movie file specified by the `myFileSpec` parameter. If your QuickTime VR movie is a panorama, the

flattened movie file includes a quarter size, blurred JPEG, compressed preview of the panorama image.

Note
The constants `MovieExportType` and `MovieFileType` used in Listing 5-3 are defined in header files `QuickTimeComponents.h` and `Movies.h` respectively and are defined as `'spit'` and `'MooV'`. ◆

You can present users with the QTVR Flattener's own dialog box to allow them to choose options such as how to compress the preview image or to select a separate preview image file. Use the following code to show the dialog box:

```
err = MovieExportDoUserDialog (qtvrExport, myQTVRMovie, nil, 0, 0,
&cancel);
```

If the user cancels the dialog box, then the Boolean cancel will be set to `true`.

If you do not want to present the user with the flattener's dialog box, you can communicate directly with the component by using the `MovieExportSetSettingsFromAtomContainer` routine as described in the following paragraphs.

If you want to specify a preview image other than the default, you need to create a special atom container and then call `MovieExportSetSettingsFromAtomContainer` before calling `MovieExportToFile`. You can specify how to compress the image, what resolution to use, and you can even specify your own preview image file to be used. The atom container you pass in can have various atoms that specify certain export options. These atoms must all be children of a flattener settings parent atom.

The preview resolution atom is a 16-bit value that allows you to specify the resolution of the preview image. This value, which defaults to `kQTVRQuarterRes`, indicates how much to reduce the preview image.

The blur preview atom is a Boolean value that indicates whether to blur the image before compressing. Blurring usually results in a much more highly compressed image. The default value is `true`.

The create preview atom is a Boolean value that indicates whether a preview image should be created. The default value is `true`.

The import preview atom is a Boolean value that is used to indicate that the preview image should be imported from an external file rather than generated

from the image in the panorama file itself. This allows you to have any image you want as the preview for the panorama. You can specify which file to use by also including the import specification atom, which is an `FSSpec` data structure that identifies the image file. If you do not include this atom, then the flattener presents the user with a dialog box asking the user to select a file. The default for import preview is `false`. If an import file is used, the image is used at its natural size and the resolution setting is ignored.

Sample Atom Container for the QTVR Flattener

The sample code in Listing 5-4 creates an atom container and adds atoms to indicate an import preview file for the flattener to use.

Listing 5-4 Specifying a preview file for the flattener to use

```
Boolean yes = true;
QTAtomContainer exportData;
QTAtom parent;
err = QTNewAtomContainer(&exportData);
// create a parent for the other settings atoms
err = QTInsertChild (exportData, kParentAtomIsContainer,
            QTVRFlattenerParentAtomType, 1, 0, 0, nil, &parent);
// Add child atom to indicate we want to import the preview from a file
err = QTInsertChild (exportData, parent, QTVRImportPreviewAtomType, 1, 0,
            sizeof (yes), &yes, nil);
// Add child atom to tell which file to import
err = QTInsertChild (exportData, parent, QTVRImportSpecAtomType, 1, 0,
            sizeof (previewSpec), &previewSpec, nil);
// Tell the export component
MovieExportSetSettingsFromAtomContainer (qtvrExport, exportData);
```

Overriding the compression settings is a bit more complicated. You need to open a standard image compression dialog component and make calls to obtain an atom container that you can then pass to the QTVR Flattener component.

Listing 5-5 Overriding the compression settings

```
ComponentInstance sc;
QTAtomContainer compressorData;
SCSpatialSettings ss;
sc =
OpenDefaultComponent(StandardCompressionType,StandardCompressionSubType);
ss.codecType = kCinepakCodecType;
ss.codec = nil;
ss.depth = 0;
ss.spatialQuality = codecHighQuality
err = SCSetInfo(sc, scSpatialSettingsType, &ss);
err = SCGetSettingsAsAtomContainer(sc, &compressorData);
MovieExportSetSettingsFromAtomContainer (qtvrExport, compressorData);
```

New Atom Types

The following atom types are planned for a header file that will be made available in a future software development kit.

```
enum {
 QTVRFlattenerParentAtomType= FOUR_CHAR_CODE('VRWe'), // parent of settings
                                        // atoms (other than compression)
 QTVRPreviewResAtomType= FOUR_CHAR_CODE('PRes'), // preview res Int16
 QTVRImportSpecAtomType= FOUR_CHAR_CODE('ISpe'), // import file FSSpec
 QTVRCreatePreviewAtomType= FOUR_CHAR_CODE('Prev'), // Boolean
 QTVRImportPreviewAtomType= FOUR_CHAR_CODE('IPre'), // Boolean
 QTVRBlurPreviewAtomType= FOUR_CHAR_CODE('Blur')  // Boolean
};

enum {
 QTVRFlattenerType= FOUR_CHAR_CODE('vrwe')  // manufacturer type for
                                            //QTVR Flattener component
}
```

Summary of the VR World and Node Atom Types

This section includes information that pertains to both this chapter and to Chapter 4, “QuickTime VR Atom Containers.”

C Summary

Constants

Version Numbers

```
#define kQTVRMajorVersion                       (2)
#define kQTVRMinorVersion                       (0)
```

VR World Atom Types

```
enum {
    kQTVRWorldHeaderAtomType    = FOUR_CHAR_CODE('vrsc'),
    kQTVRImagingParentAtomType  = FOUR_CHAR_CODE('imgp'),
    kQTVRPanoImagingAtomType    = FOUR_CHAR_CODE('impn'),
    kQTVRObjectImagingAtomType  = FOUR_CHAR_CODE('imob'),
    kQTVRNodeParentAtomType     = FOUR_CHAR_CODE('vrnp'),
    kQTVRNodeIDAtomType         = FOUR_CHAR_CODE('vrni'),
    kQTVRNodeLocationAtomType   = FOUR_CHAR_CODE('nloc')
};
```

Node Information Atom Types

```
enum {
    kQTVRNodeHeaderAtomType     = FOUR_CHAR_CODE('ndhd'),
    kQTVRHotSpotParentAtomType  = FOUR_CHAR_CODE('hspa'),
    kQTVRHotSpotAtomType        = FOUR_CHAR_CODE('hots'),
```

```
    kQTVRHotSpotInfoAtomType    = FOUR_CHAR_CODE('hsin'),
    kQTVRLinkInfoAtomType       = FOUR_CHAR_CODE('link')
};
```

Miscellaneous Atom Types

```
enum {
    kQTVRStringAtomType             = FOUR_CHAR_CODE('vrsg'),
    kQTVRPanoSampleDataAtomType     = FOUR_CHAR_CODE('pdat'),
    kQTVRObjectInfoAtomType         = FOUR_CHAR_CODE('obji'),
    kQTVRAltImageTrackRefAtomType   = FOUR_CHAR_CODE('imtr'),
    kQTVRAltHotSpotTrackRefAtomType = FOUR_CHAR_CODE('hstr'),
    kQTVRAngleRangeAtomType         = FOUR_CHAR_CODE('arng'),
    kQTVRTrackRefArrayAtomType      = FOUR_CHAR_CODE('tref'),
    kQTVRPanConstraintAtomType      = FOUR_CHAR_CODE('pcon'),
    kQTVRTiltConstraintAtomType     = FOUR_CHAR_CODE('tcon'),
    kQTVRFOVConstraintAtomType      = FOUR_CHAR_CODE('fcon')
};
```

Track Reference Types

```
enum {
    kQTVRImageTrackRefType      = FOUR_CHAR_CODE('imgt'),
    kQTVRHotSpotTrackRefType    = FOUR_CHAR_CODE('hott')
};
```

Imaging Property Valid Flags

```
enum {
    kQTVRValidCorrection                        = 1 << 0,
    kQTVRValidQuality                           = 1 << 1,
    kQTVRValidDirectDraw                        = 1 << 2,
    kQTVRValidFirstExtraProperty                = 1 << 3
};
```

Link Hot Spot Valid Bits

```
enum {
    kQTVRValidPan                               = 1 << 0,
    kQTVRValidTilt                              = 1 << 1,
```

```
    kQTVRValidFOV                           = 1 << 2,
    kQTVRValidViewCenter                    = 1 << 3
};
```

Animation Settings

```
enum QTVRAnimationSettings {
    kQTVRObjectAnimateViewFramesOn          = (1 << 0),
    kQTVRObjectPalindromeViewFramesOn       = (1 << 1),
    kQTVRObjectStartFirstViewFrameOn        = (1 << 2),
    kQTVRObjectAnimateViewsOn               = (1 << 3),
    kQTVRObjectPalindromeViewsOn            = (1 << 4),
    kQTVRObjectSyncViewToFrameRate          = (1 << 5),
    kQTVRObjectDontLoopViewFramesOn         = (1 << 6),
    kQTVRObjectPlayEveryViewFrameOn         = (1 << 7)
};
```

Control Settings

```
enum QTVRControlSettings {
    kQTVRObjectWrapPanOn                    = (1 << 0),
    kQTVRObjectWrapTiltOn                   = (1 << 1),
    kQTVRObjectCanZoomOn                    = (1 << 2),
    kQTVRObjectReverseHControlOn            = (1 << 3),
    kQTVRObjectReverseVControlOn            = (1 << 4),
    kQTVRObjectSwapHVControlOn              = (1 << 5),
    kQTVRObjectTranslationOn                = (1 << 6)
};
```

Controller Subtype and ID

```
enum {
    kQTControllerType                       = FOUR_CHAR_CODE('ctyp').
    kQTControllerID                         = 1
};
```

Object Controller Types

```
enum ObjectUITypes {
    kGrabberScrollerUI                      = 1,
    kOldJoyStickUI                          = 2,
```

```
    kJoystickUI                             = 3,
    kGrabberUI                              = 4,
    kAbsoluteUI                             = 5
};
```

Node Location Flag

```
enum {
    kQTVRSameFile                           = 0
};
```

Panorama Sample Flag

```
enum {
    kQTVRPanoFlagHorizontal                 = 1 << 0
};
```

Data Types

```
typedef float                                   Float32;
```

Sample Description Header Structure

```
typedef struct QTVRSampleDescription {
    UInt32                              size;
    UInt32                              type;
    UInt32                              reserved1;
    UInt16                              reserved2;
    UInt16                              dataRefIndex;
    UInt32                              data;
} QTVRSampleDescription, *QTVRSampleDescriptionPtr, **QTVRSampleDescriptionHandle;
```

String Atom Structure

```
typedef struct VRStringAtom {
    UInt16                              stringUsage;
    UInt16                              stringLength;
    unsigned char                       string[4];
} VRStringAtom, *VRStringAtomPtr;
```

VR World Header Atom Structure

```
typedef struct VRWorldHeaderAtom {
    UInt16                          majorVersion;
    UInt16                          minorVersion;
    QTAtomID                        nameAtomID;
    UInt32                          defaultNodeID;
    UInt32                          vrWorldFlags;
    UInt32                          reserved1;
    UInt32                          reserved2;
} VRWorldHeaderAtom, *VRWorldHeaderAtomPtr;
```

Panorama-Imaging Atom Structure

```
typedef struct VRPanoImagingAtom {
    UInt16                          majorVersion;
    UInt16                          minorVersion;
    UInt32                          imagingMode;
    UInt32                          imagingValidFlags;
    UInt32                          correction;
    UInt32                          quality;
    UInt32                          directDraw;
    UInt32                          imagingProperties[6];
    UInt32                          reserved1;
    UInt32                          reserved2;
} VRPanoImagingAtom, *VRPanoImagingAtomPtr;
```

Node Location Atom Structure

```
typedef struct VRNodeLocationAtom {
    UInt16                          majorVersion;
    UInt16                          minorVersion;
    OSType                          nodeType;
    UInt32                          locationFlags;
    UInt32                          locationData;
    UInt32                          reserved1;
    UInt32                          reserved2;
} VRNodeLocationAtom, *VRNodeLocationAtomPtr;
```

Node Header Atom Structure

```
typedef struct VRNodeHeaderAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    OSType                              nodeType;
    QTAtomID                            nodeID;
    QTAtomID                            nameAtomID;
    QTAtomID                            commentAtomID;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRNodeHeaderAtom, *VRNodeHeaderAtomPtr;
```

Hot Spot Information Atom Structure

```
typedef struct VRHotSpotInfoAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    OSType                              hotSpotType;
    QTAtomID                            nameAtomID;
    QTAtomID                            commentAtomID;
    SInt32                              cursorID[3];
    Float32                             bestPan;
    Float32                             bestTilt;
    Float32                             bestFOV;
    FloatPoint                          bestViewCenter;
    Rect                                hotSpotRect;
    UInt32                              flags;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRHotSpotInfoAtom, *VRHotSpotInfoAtomPtr;
```

Link Hot Spot Atom Structure

```
typedef struct VRLinkHotSpotAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    UInt32                              toNodeID;
    UInt32                              fromValidFlags;
    Float32                             fromPan;
    Float32                             fromTilt;
```

```
    Float32                             fromFOV;
    FloatPoint                          fromViewCenter;
    UInt32                              toValidFlags;
    Float32                             toPan;
    Float32                             toTilt;
    Float32                             toFOV;
    FloatPoint                          toViewCenter;
    Float32                             distance;
    UInt32                              flags;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRLinkHotSpotAtom, *VRLinkHotSpotAtomPtr;
```

Angle Range Atom Structure

```
typedef struct VRAngleRangeAtom {
    Float32                             minimumAngle;
    Float32                             maximumAngle;
} VRAngleRangeAtom, *VRAngleRangeAtomPtr;
```

Panorama Sample Atom Structure

```
typedef struct VRPanoSampleAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    UInt32                              imageRefTrackIndex;
    UInt32                              hotSpotRefTrackIndex;
    Float32                             minPan;
    Float32                             maxPan;
    Float32                             minTilt;
    Float32                             maxTilt;
    Float32                             minFieldOfView;
    Float32                             maxFieldOfView;
    Float32                             defaultPan;
    Float32                             defaultTilt;
    Float32                             defaultFieldOfView;
    UInt32                              imageSizeX;
    UInt32                              imageSizeY;
    UInt16                              imageNumFramesX;
    UInt16                              imageNumFramesY;
    UInt32                              hotSpotSizeX;
```

```
    UInt32                              hotSpotSizeY;
    UInt16                              hotSpotNumFramesX;
    UInt16                              hotSpotNumFramesY;
    UInt32                              flags;
    UInt32                              reserved1;
    UInt32                              reserved2;
} VRPanoSampleAtom, *VRPanoSampleAtomPtr;
```

Object Sample Atom Structure

```
typedef struct VRObjectSampleAtom {
    UInt16                              majorVersion;
    UInt16                              minorVersion;
    UInt16                              movieType;
    UInt16                              viewStateCount;
    UInt16                              defaultViewState;
    UInt16                              mouseDownViewState;
    UInt32                              viewDuration;
    UInt32                              columns;
    UInt32                              rows;
    Float32                             mouseMotionScale;
    Float32                             minPan;
    Float32                             maxPan;
    Float32                             defaultPan;
    Float32                             minTilt;
    Float32                             maxTilt;
    Float32                             defaultTilt;
    Float32                             minFieldOfView;
    Float32                             fieldOfView;
    Float32                             defaultFieldOfView;
    Float32                             defaultViewCenterH;
    Float32                             defaultViewCenterV;
    Float32                             viewRate;
    Float32                             frameRate;
    UInt32                              animationSettings;
    UInt32                              controlSettings;
} VRObjectSampleAtom, *VRObjectSampleAtomPtr;
```

Track Reference Entry Structure

```
struct QTVRTrackRefEntry {
    UInt32                      trackRefType;
    UInt16                      trackResolution;
    UInt32                      trackRefIndex;
};
typedef struct QTVRTrackRefEntry QTVRTrackRefEntry;
```

APPENDIX A

QuickTime VR Cursors

This appendix lists the cursors that QuickTime VR uses for interactive feedback. The cursors are categorized according to the way they are used. The categories are:

- Hot spot cursors
- Navigation cursors
- Manipulation cursors
  - Panning interface cursors
  - Grabber interface cursors
  - Spinner interface cursors
  - Joystick interface cursors
  - Pointer interface cursor

Hot Spot Cursors

Hot spot cursors have ID values in the range –32000 to –31501.

Table A-1 Hot spot cursors

Cursor ID	Cursor icon	Cursor description
–32000		Mouse over link (`'link'`) hot spot cursor.
–32199		Mouse down on link (`'link'`) hot spot cursor.
–32198		Mouse-up cursor while link (`'link'`) hot spot is executing.
–32197		Mouse over undefined object, that is, any hot spot of a type not covered by other hot spot cursors.

Table A-1 Hot spot cursors (continued)

Cursor ID	Cursor icon	Cursor description
–32196		Mouse down on undefined object, that is, any hot spot of a type not covered by other hot spot cursors.
–32195		Mouse-up cursor while undefined object hot spot (that is, any hot spot of a type not covered by other hot spot cursors) is executing.
–32194		Mouse over URL (`'url '`) hot spot cursor
–32193		Mouse down on URL (`'url '`) hot spot cursor
–32192		Mouse-up cursor while URL (`'url '`) hot spot is executing

Navigation Cursors

Navigation cursors have ID values in the range –31500 to –31001.

Table A-2 Navigation cursors

Cursor ID	Cursor icon	Cursor description
–31500		Zooming in cursor.
–31499		Zooming out cursor.
–31498		Conflicting zoom keys cursor.
–31497		Zooming in cursor, with zooming constrained.

Table A-2 Navigation cursors (continued)

Cursor ID	Cursor icon	Cursor description
–31496		Zooming out cursor, with zooming constrained.
–31495		Translate on cursor.
–31494		Translate on, but unable to translate cursor.

Manipulation Cursors

The manipulation cursors are listed in the following categories:

- Panning interface cursors
- Grabber interface cursors
- Spinner interface cursors
- Joystick interface cursors
- Pointer interface cursor

Panning Interface Cursors

Panning interface cursors have ID values in the range –19000 to –18501.

Table A-3 Panning interface cursors

Cursor ID	Cursor icon	Cursor description
–19000		Panning interface mouse-up cursor.
–18999		Mouse centered, waiting to pan or tilt cursor.
–18998		Pan left cursor.
–18997		Pan left cursor, with panning constrained.
–18996		Pan right cursor.
–18995		Pan right cursor, with panning constrained.
–18994		Pan down cursor.
–18993		Pan down cursor, with panning constrained.
–18992		Pan down-left cursor.
–18991		Pan down-left cursor, with panning down constrained.
–18990		Pan down-left cursor, with panning left constrained.
–18989		Pan down-left cursor, with both panning down and panning left constrained.
–18988		Pan down-right cursor.
–18987		Pan down-right cursor, with panning down constrained.

Table A-3 Panning interface cursors (continued)

Cursor ID	Cursor icon	Cursor description
–18986		Pan down-right cursor, with panning right constrained.
–18985		Pan down-right cursor, with both panning down and panning right constrained.
–18984		Pan up cursor.
–18983		Pan up cursor, with panning constrained.
–18982		Pan up-left cursor.
–18981		Pan up-left cursor, with panning up constrained.
–18980		Pan up-left cursor, with panning left constrained.
–18979		Pan up-left cursor, with both panning up and panning left constrained.
–18978		Pan up-right cursor.
–18977		Pan up-right cursor, with panning up constrained.
–18976		Pan up-right cursor, with panning right constrained.
–18975		Pan up-right cursor, with both panning up and panning right constrained.

Grabber Interface Cursors

Grabber interface cursors have ID values in the range –18500 to –18001.

Table A-4 Grabber interface cursors

Cursor ID	Cursor icon	Cursor description
–18500		Mouse-up cursor when over the central part of the movie.
–18499		Mouse-down cursor when over the central part of the movie.

Spinner Interface Cursors

Spinner interface cursors have ID values in the range –18000 to –17501.

Table A-5 Spinner interface cursors

Cursor ID	Cursor icon	Cursor description
–18000		Inside top border cursor, both mouse up and mouse down.
–17999		Inside bottom border cursor, both mouse up and mouse down.
–17998		Inside left border cursor, both mouse up and mouse down, when tilt angle is between 78˚ and 90˚.
–17997		Inside right border cursor, both mouse up and mouse down, when tilt angle is between 78˚ and 90˚.
–17996		Inside left border cursor, both mouse up and mouse down, when tilt angle is between 56˚ and 78˚.
–17995		Inside right border cursor, both mouse up and mouse down, when tilt angle is between 56˚ and 78˚.
–17994		Inside left border cursor, both mouse up and mouse down, when tilt angle is between 34˚ and 56˚.

Table A-5 Spinner interface cursors (continued)

Cursor ID	Cursor icon	Cursor description
–17993		Inside right border cursor, both mouse up and mouse down, when tilt angle is between 34˚ and 56˚.
–17992		Inside left border cursor, both mouse up and mouse down, when tilt angle is between 11˚ and 34˚.
–17991		Inside right border cursor, both mouse up and mouse down, when tilt angle is between 11˚ and 34˚.
–17990		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –11˚ and 11˚.
–17989		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –11˚ and 11˚.
–17988		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –34˚ and –11˚.
–17987		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –34˚ and –11˚.
–17986		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –56˚ and –34˚.
–17985		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –56˚ and –34˚.
–17984		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –78˚ and –56˚.
–17983		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –78˚ and –56˚.
–17982		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –90˚ and –78˚.
–17981		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –90˚ and –78˚.
–17980		Inside top border cursor, both mouse up and mouse down; object cannot turn up.
–17979		Inside bottom border cursor, both mouse up and mouse down; object cannot turn down.

Table A-5 Spinner interface cursors (continued)

Cursor ID	Cursor icon	Cursor description
–17978		Inside left border cursor, both mouse up and mouse down, when tilt angle is between 78˚ and 90˚; object cannot turn left.
–17977		Inside right border cursor, both mouse up and mouse down, when tilt angle is between 78˚ and 90˚; object cannot turn right.
–17976		Inside left border cursor, both mouse up and mouse down, when tilt angle is between 56˚ and 78˚; object cannot turn left.
–17975		Inside right border cursor, both mouse up and mouse down, when tilt angle is between 56˚ and 78˚; object cannot turn right.
–17974		Inside left border cursor, both mouse up and mouse down, when tilt angle is between 34˚ and 56˚; object cannot turn left.
–17973		Inside right border cursor, both mouse up and mouse down, when tilt angle is between 34˚ and 56˚; object cannot turn right.
–17972		Inside left border cursor, both mouse up and mouse down, when tilt angle is between 11˚ and 34˚; object cannot turn left.
–17971		Inside right border cursor, both mouse up and mouse down, when tilt angle is between 11˚ and 34˚; object cannot turn right.
–17970		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –11˚ and 11˚; object cannot turn left.
–17969		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –11˚ and 11˚; object cannot turn right.
–17968		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –34˚ and –11˚; object cannot turn left.
–17967		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –34˚ and –11˚; object cannot turn right.

Table A-5 Spinner interface cursors (continued)

Cursor ID	Cursor icon	Cursor description
–17966		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –56˚ and –34˚; object cannot turn left.
–17965		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –56˚ and –34˚; object cannot turn right.
–17964		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –78˚ and –56˚; object cannot turn left.
–17963		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –78˚ and –56˚; object cannot turn right.
–17962		Inside left border cursor, both mouse up and mouse down, when tilt angle is between –90˚ and –78˚; object cannot turn left.
–17961		Inside right border cursor, both mouse up and mouse down, when tilt angle is between –90˚ and –78˚; object cannot turn right.

Joystick Interface Cursors

Joystick interface cursors have ID values in the range –17500 to –17001.

Table A-6 Joystick interface cursors

Cursor ID	Cursor icon	Cursor description
–17500		Mouse up, joystick center cursor.
–17499		Mouse up, cursor left of joystick center.
–17498		Mouse up, cursor right of joystick center.

Table A-6 Joystick interface cursors (continued)

Cursor ID	Cursor icon	Cursor description
–17497		Mouse up, cursor bottom of joystick center.
–17496		Mouse up, cursor bottom left of joystick center.
–17495		Mouse up, cursor bottom right of joystick center.
–17494		Mouse up, cursor top of joystick center.
–17493		Mouse up, cursor top left of joystick center.
–17492		Mouse up, cursor top right of joystick center.
–17491		Mouse down, joystick center cursor.
–17490		Mouse down, cursor left of joystick center.
–17489		Mouse down, cursor right of joystick center.
–17488		Mouse down, cursor bottom of joystick center.
–17487		Mouse down, cursor bottom left of joystick center.
–17486		Mouse down, cursor bottom right of joystick center.
–17485		Mouse down, cursor top of joystick center.
–17484		Mouse down, cursor top left of joystick center.
–17483		Mouse down, cursor top right of joystick center.

Pointer Interface Cursor

Pointer interface cursors have ID values in the range –17000 to –16501.

Table A-7 Pointer interface cursor

Cursor ID	Cursor icon	Cursor description
–17000		Pointer interface cursor.

Compatibility With QuickTime VR 2.0

The cursor ID values shown here are new with QuickTime VR 2.1. For cursor IDs used with QuickTime VR 2.0, developers should refer to the old documentation.

When you call `QTVRReplaceCursor()` with an old cursor ID, the ID value will be updated in the `CursorRecord` structure on return, to correspond to the ID that was actually replaced—the cursor's new ID.

When you call `QTVRReplaceCursor()` with an old ID value that was never used, the function may return a paramErr even though no error was returned by QuickTime VR 2.0. Similarly, QuickTime VR returns a paramErr for any cursor ID that is not used in the current movie: for example, a cursor ID used only by object-type nodes when the current movie consists only of panoramic nodes. From this point of view, the cursor ID values that no longer exist may be thought of as belonging to a node type that doesn't exist in any QuickTime VR movie.

APPENDIX A

QuickTime VR Cursors

Glossary

action See **movie controller action.**

action filter function An application-defined function that is called by a movie controller component each time the component receives a movie controller action. Your filter function can handle the action itself or refer it back to the movie controller component for processing**.**

angular unit A unit for measuring angles. For the QuickTime VR Manager, the default angular unit is degrees. Compare **degree, radian.**

animation See **frame animation, view animation.**

animation setting A value that specifies characteristics of an object node while it is playing an animation. For example, one animation setting determines whether an object node animation is looped. Compare **control setting.**

antialiasing The smoothing of jagged edges on a displayed shaped by modifying the transparencies of individual pixels along the shape's edge.

API See **application programming interface.**

application programming interface (API) The total set of constants, data structures, routines, and other programming elements that allow developers to use some part of the system software.

area of interest structure A data structure that delineates a rectangular area in the back buffer. Defined by the `AreaOfInterest` data type.

atom The basic unit of data in a movie resource. Defined by the `QTAtom` data type. There are a number of different atom types, including movie atoms, track atoms, and media atoms. There are two varieties of atoms: container atoms, which contain other atoms, and leaf atoms, which do not contain any other atoms.

atom container An opaque object that contains one or more atoms. Defined by the `QTAtomContainer` data type.

author (v) To create a multimedia title, such as a QuickTime VR movie. (n) The person or persons who creates a multimedia title.

autoscrolling The scrolling that occurs when the mouse button is being held down and the user is zooming.

back buffer An image buffer maintained by QuickTime VR for panoramic nodes that contains an image that has not yet been corrected. Compare **prescreen buffer.**

back buffer imaging procedure An application-defined procedure that is executed at established times; you use this procedure to access the back buffer. Compare **prescreen buffer imaging completion procedure.**

base resolution The highest resolution image present in a node.

cache buffer See **back buffer.**

constraint See **viewing constraint.**

container atom A QuickTime atom that contains other atoms, possibly including other container atoms. Examples of container atoms are track atoms and edit atoms. Compare **leaf atom.**

controller See **movie controller component, QuickTime VR movie controller.**

controller bar A rectangular region associated with a QuickTime VR movie that displays the movie's controls.

controller bar key A key that, when pressed, activates a control in the controller bar. For example, the translation key is a controller bar key.

control setting A value that specifies whether an object can wrap during panning and tilting, as well as other features of the node. Compare **animation setting.**

correct To undo the cylindrical (or other) projection of a panoramic image so that it appears as it was originally captured or rendered. Also called *warp.*

correction mode The kind of warping correction to be applied when imaging a panoramic view.

current node The node in a scene that is currently being viewed on the screen.

cursor record A data structure that indicates which cursor to replace and its replacement cursor. Defined by the `CursorRecord` data type.

custom hot spot A hot spot of a type defined by a third-party developer.

custom hot spot cursor A custom cursor that is displayed during interaction with a hot spot.

cylindrical projection A method of projecting a panorama onto the surface of a cylinder.

default node The node in a scene that is displayed when the scene is first opened.

degree An angular unit equal to 1/360 of a complete circle. Indicated by the sign °. Compare **radian.**

destination node The node to which a link hot spot is linked.

dicing The process of transforming a panoramic image into individual frames in a panorama image track.

enviromapping The process of projecting an image onto the inside of a shape, such as a sphere or cylinder.

field of view The horizontal or vertical angular expanse visible through a camera.

field-of-view constraint atom An atom in a QuickTime VR file that contains information about the initial vertical field of view constraints for a panorama. Compare **pan constraint atom, tilt constraint atom.**

floating-point point structure A data structure used to specify a point in a panorama or object. Defined by the `FloatPoint` data type.

FOV See **field of view.**

frame animation An animation through all frames of a particular view. Compare **view animation.**

frame animation rate See **frame rate.**

frame differencing A form of temporal compression that involves examining redundancies between adjacent frames in a moving image sequence.

frame rate The rate at which the frame animation in a view is played. Compare **view rate.**

front buffer See **prescreen buffer.**

full panorama A 360-degree panorama**.** Compare **partial panorama.**

go-back button A button in the controller bar for panoramas that allows the user to return to the previous node.

horizontal field of view The horizontal angular expanse visible through a camera. Compare **vertical field of view.**

hot spot An area in a QuickTime VR movie that permits user interaction.

hot spot action selector A value passed to a mouse over hot spot procedure that indicates whether the cursor has entered, is in, or has left the hot spot.

hot spot atom An atom in a QuickTime VR file that contains information about a hot spot. A hot spot atom is always a child of a hot spot parent atom.

hot spot callback routine See **mouse over hot spot procedure.**

hot spot display button A button in the controller bar that allows the user to highlight the visible hot spots.

hot spot ID An unsigned long integer that uniquely identifies a hot spot.

hot spot image track A video track in a QuickTime VR file that contains colored regions depicting the hot spots. For objects, an image in the hot spot image track must be synchronized in time to the corresponding image in the object image track.

hot spot parent atom An atom in a QuickTime VR file that contains one or more hot spot atoms and their children.

hysteresis See **mouse-click hysteresis.**

image See **movie image.**

Image Compression Manager The part of QuickTime that provides image compression and decompression services.

image track See **low-resolution image track, object image track, panorama image track.**

imaging atom An atom in a QuickTime VR file that describes the default imaging characteristics for one type of node in a scene. Imaging atoms are contained in an imaging parent atom. See also **panorama-imaging atom.**

imaging mode A mode or state of a panoramic node that determines the kind of drawing that is to be performed for the node.

imaging parent atom An atom in a QuickTime VR file that contains one or more imaging atoms.

imaging property A property of a node that determines a drawing characteristic for that node (such as the image quality).

instance See **QuickTime VR movie instance.**

interaction property A property of a QuickTime VR movie whose value determines a specific mode of user interaction.

intercept procedure A function, installed by a call to `QTVRInstallInterceptProc`, that is executed instead of (or in addition to) a QuickTime VR Manager function.

intercept selector A constant that indicates which QuickTime VR Manager function to intercept. Defined by the `QTVRProcSelector` enumerated constants.

intercept structure A data structure used to pass information to an intercept procedure. Defined by the `QTVRInterceptRecord` data type.

key node An important node in a scene.

label display area An area in the controller bar in which information can be displayed. For instance, when the cursor is over one of the buttons, the button's name appears in the label display area.

leaf atom A QuickTime atom that contains no other atoms. Compare **container atom.**

limit See **viewing limit.**

link A connection between two nodes in a QuickTime VR movie.

link hot spot A hot spot that allows a user to navigate from one node in a scene to another.

lossless compression A compression scheme that preserves all of the original data.

low-resolution image track A video track that contains a diced low-resolution version of a panoramic image.

mouse-click hysteresis The distance, in pixels, from the location of a mouse-down event to the limit in which the cursor is considered not to have moved.

mouse-click timeout The number of ticks after which a mouse click times out and is automatically switched from a hot spot selection into a pan.

mouse-down tracking The process of tracking the state of the mouse button when it is clicked in a QuickTime VR movie. Compare **mouse-over tracking.**

mouse-motion scale A value that indicates the number of degrees or radians that an object or panorama is panned or tilted when the cursor is dragged the entire width of the VR movie image.

mouse over hot spot procedure An application-defined procedure that is executed whenever the cursor moves over, remains above, or leaves a hot spot.

mouse-over tracking The process of tracking the location of the cursor when it is in a QuickTime VR movie and changing the shape of the cursor as appropriate. Compare **mouse-down tracking.**

movie A set of time-based data that is managed by the Movie Toolbox. A QuickTime movie may contain sound, video, animation, laboratory results, financial data, or a combination of any of these types of time-based data. A QuickTime movie contains one or more tracks; each track represents a single data stream in the movie.

movie author See **author.**

movie controller See **movie controller component, QuickTime VR movie controller.**

movie controller action A constant (of type `mcAction`) used by QuickTime movie controller components in the `MCDoAction` function. Applications that include action filters can receive any of these actions.

movie controller component A component that manages the user interface for playing and editing movies.

movie image The image currently displayed in a QuickTime VR movie.

movie instance See **QuickTime VR movie instance.**

multinode movie A QuickTime VR movie that includes more than one node. Compare **single-node movie.**

navigable node See **object node.**

node A position in a scene at which you view an object or a panorama.

node-entering procedure An application-defined procedure that is executed each time the user enters a node. Compare **node-leaving procedure.**

node header atom An atom in a QuickTime VR file that contains the type and ID of a node, as well as the node name and any node comments.

node ID An unsigned long integer that uniquely identifies a node in a movie.

node ID atom An atom in a QuickTime VR file that specifies the ID of a node in the scene. Each node ID atom has a single child, which is a node location atom.

node information atom container An atom container that contains information about a particular node in a scene (for example, the hot spots and default characteristics of the node).

node-leaving procedure An application-defined procedure that is executed each time the user leaves a node. Compare **node-entering procedure.**

node location atom An atom in a QuickTime VR file that specifies the type and location of a node.

node parent atom An atom in a QuickTime VR file that contains one or more node ID atoms.

nudge To move to the next available view in a specified direction.

nudge control A constant that indicates the direction in which to nudge.

object (1) The real or synthetic object that is captured photographically or rendered by computer to create an object node. (2) See **object node.**

object image array A two-dimensional array of images that represent all possible views of an object.

object image track A video track in a QuickTime VR movie file that contains, in sequential order, the views of an object**.**

object node A type of node that allows a user to manipulate a photographically captured or computer-rendered object. Compare **panoramic node.**

object sample atom An atom in a QuickTime VR file that describes a single object.

object track A track that contains information about the object nodes in a scene. Compare **panorama track.**

object view A single set of values for the field of view, view center, pan angle, and tilt angle for an object node. Compare **panorama view.**

palindrome looping A type of looping in which the frames in a view duration are played forward, then backward, then forward, and so on.

pan To move a camera or point of view horizontally. Compare **tilt.**

pan angle The angle of pan. Pan angles are measured in radians or degrees, where positive values pan to the left. Compare **tilt angle.**

pan constraint atom An atom in a QuickTime VR file that contains information about the initial pan angle constraints for a panorama. Compare **field-of-view constraint atom, tilt constraint atom.**

panning speed The relative speed of panning and tilting. This speed should be from 1 (the slowest speed) through 10 (the fastest speed); the default panning and tilting speed is 5.

panorama (1) The real or synthetic expanse that is captured photographically or rendered by computer to create a panoramic node. (2) See **panoramic node.**

panorama buffer See **back buffer.**

panorama data atom See **panorama sample atom.**

panorama image track A video track in a QuickTime VR movie file that contains the diced frames of the original panoramic image**.**

panorama-imaging atom An atom in a QuickTime VR file that describes the default imaging characteristics for the panoramic nodes in a scene.

panorama sample atom An atom in a QuickTime VR file that describes a single panorama, including information about the default viewing angles and the source panoramic image.

panorama track A track that contains information about the panoramic nodes in a scene. Compare **object track.**

panorama view A single set of values for the field of view, pan angle, and tilt angle for a panoramic node. Compare **object view.**

panoramic image A cylindrical band of blended photographed or rendered images. Also called a *panoramic PICT.*

panoramic node A type of node that allows a user to view a panorama. Compare **object node.**

panoramic PICT See **panoramic image.**

partial panorama A panorama that is less than 360 degrees. Compare **full panorama.**

play flag See **animation setting, control setting.**

play rate The rate (and direction) at which a movie is played.

play setting See **animation setting, control setting.**

prescreen buffer An image buffer maintained by QuickTime VR for panoramic nodes that contains the corrected image. Also called the *front buffer*. Compare **back buffer.**

prescreen buffer imaging completion procedure An application-defined procedure that is executed each time QuickTime VR is finished drawing an image into the prescreen buffer. Compare **back buffer imaging procedure.**

procedure selector See **intercept selector.**

QTVR See **QuickTime VR.**

QTVR instance See **QuickTime VR movie instance.**

QTVR movie See **QuickTime VR movie.**

QTVR track A track in a QuickTime VR movie that contains a list of the nodes in the scene and other information about the scene.

QuickTime A part of the system software for Macintosh and other computers that an application can use to control time-based data, such as video or audio data.

QuickTime atom See **atom**.

QuickTime movie A set of time-based data that can be displayed and manipulated using QuickTime.

QuickTime VR An extension of the QuickTime technology developed by Apple Computer, Inc. that allows users to interactively explore and examine photorealistic, three-dimensional virtual worlds.

QuickTime VR file format The format of the movie files that contain QuickTime VR movies.

QuickTime VR Manager The part of the system software for Macintosh and other computers that an application can use to control QuickTime VR movies.

QuickTime VR movie A collection of object and panoramic nodes.

QuickTime VR movie controller The movie controller component that manages QuickTime VR movies. See also **controller bar.**

QuickTime VR movie instance An instance of a QuickTime VR movie. Defined by the `QTVRInstance` data type.

radian An angular unit defined by the circular angle whose radius is equal to the subtended circular arc. Abbreviated rad. A radian is 57.2958 degrees. Compare **degree.**

scale See **zoom.**

scene (1) A site or location at which you want to make a QuickTime VR movie. (2) A collection of one or more nodes.

scene header atom See **VR world header atom.**

sibling atom An atom that has the same parent as some other atom (that is, is contained in the same container).

single-node movie A QuickTime VR movie that includes just one node. Compare **multinode movie.**

spatial compression Image compression that is performed in the context of a single frame. This compression technique takes advantage of redundancy in the image to reduce the amount of data that is required to accurately represent the image. Compare **temporal compression.**

speaker button A button in the controller bar that allows the user to adjust the sound volume.

string atom An atom in a QuickTime VR file that contains a string.

swing transition A smooth motion from one view in a node to another view in the same node.

synthetic Not real, as for example the objects in a computer-generated three-dimensional model.

temporal compression Image compression that is performed between frames in a sequence. This compression technique takes advantage of redundancy between adjacent frames in a sequence to reduce the amount of data that is required to accurately represent each frame in the sequence. Sequences that have been temporally compressed typically contain key frames at regular intervals. Compare **spatial compression.**

tilt To move a camera or point of view vertically. Compare **pan.**

tilt angle The angle of tilt. Tilt angles are measured in radians or degrees, where positive values tilt up. Compare **pan angle.**

tilt constraint atom An atom in a QuickTime VR file that contains information about the initial tilt angle constraints for a panorama. Compare **field-of-view constraint atom, pan constraint atom.**

tilting speed The relative speed of panning and tilting. This speed should be from 1 (the slowest speed) through 10 (the fastest speed); the default panning and tilting speed is 5.

timeout See **mouse-click timeout.**

transition The movement between two items in a movie, such as from one view in a node to another view in the same node, or from one node to another.

transition effect Any special visual effect associated with a transition.

transition property A property of a VR movie whose value determines a specific kind of transition effect.

translate To reposition an object by changing the current view center.

translate mode An interaction property for objects whose value determines whether dragging the mouse causes an object to be translated (`true`) or to be panned or tilted (`false`).

translate mode button A button in the controller bar that allows the user to enable or disable translate mode.

translation key The key on a keyboard that the user can hold down to enable translate mode.

Universal Resource Locator (URL) An address of a page on the World Wide Web.

URL See **Universal Resource Locator.**

URL hot spot A hot spot that is associated with a URL.

user data Auxiliary data that your application can store in a QuickTime movie, track, or media structure. The user data is stored in a user data list; items in the list are referred to as *user data items*. Examples of user data include a copyright, date of creation, name of a movie's director, and special hardware and software requirements.

user data item A single element in a user data list.

user data list The collection of user data for a QuickTime movie, track, or media structure.

vertical field of view The vertical angular expanse visible through a camera. Compare **horizontal field of view.**

view See **object view, panorama view.**

view angle See **pan angle, tilt angle.**

view animation An animation through all views in the current row of a particular object. Compare **frame animation.**

view animation rate See **view rate.**

view array See **object image array.**

view center The pixel in the image of a view that appears at the center of an object's bounding box.

view duration The amount of time in an object node's video track that is occupied by a particular view of an object.

viewing constraint A limit on the current viewing characteristics (pan angle, tilt angle, or field of view) for a panorama or object. A constraint is imposed at runtime and must always lie in the node's viewing limits. Compare **viewing limit.**

viewing constraint atom An atom in a QuickTime VR file that contains information about the initial viewing constraints for a panorama. Compare **field-of-view constraint atom, pan constraint atom, tilt constraint atom.**

viewing limit A physical limit on the allowable viewing characteristics (pan angle, tilt angle, or field of view) for a panorama or object. A viewing limit is imposed by the panoramic or object data stored in the movie file. Compare **viewing constraint.**

view rate The rate at which the view animation in an object is played. Compare **frame rate.**

view setting See **control setting.**

view state A state that selects an alternate set of images for an object's views. For instance, holding down the mouse button might change an object from one view state to another, causing a different set of images to be displayed.

virtual reality (VR) The experience of exploring and interacting with a spatial environment using a computer.

virtual world A spatial environment that can by explored and interacted with using a computer.

VR See **QuickTime VR, virtual reality.**

VR movie See **QuickTime VR movie.**

VR movie instance See **QuickTime VR movie instance.**

VR object See **object node.**

VR panorama See **panoramic node.**

VR world (1) See **virtual world.** (2) A data structure that contains general information about a QuickTime VR movie. A movie's VR world is contained in the QTVR track's sample description header, defined by the `QTVRSampleDescription` data type.

VR world header atom An atom in a QuickTime VR file that contains the name of the scene and the default node ID to be used when the file is first opened.

warp See **correct.**

warping correction mode See **correction mode.**

warp space buffer See **back buffer.**

zoom To enlarge or reduce the magnification of an image while maintaining the current point of interest.

zoom-in button A button in the controller bar that allows the user to zoom in.

zooming speed The relative speed of zooming in and out. This speed should be from 1 (the slowest speed) through 10 (the fastest speed); the default zooming speed is 5.

zoom-out button A button in the controller bar that allows the user to zoom out.

Index

E

F

G

H

I, J

K

L

M

N

O

P

Q

R

S

T

U

V

W, X, Y

Z

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITERS
Tim Monroe, Allen Watson

ILLUSTRATOR
Deb Dennis

DEVELOPMENTAL EDITORS
Donna S. Lee, Beverly McGuire

PRODUCTION EDITOR
Gerri Gray

PROJECT MANAGER
Michael Hinkson

Special thanks to Ken Doyle III and Bryce Wolfson