QuickTime Toolkit

ADVANCED MOVIE PLAYBACK AND MEDIA TYPES | VOLUME TWO

Apple

CD included

MORGAN KAUFMANN

QuickTime Developer Series

Buried inside QuickTime are a host of powerful tools for creating, delivering, and playing digital media. The official QuickTime documentation explains "what" each API function does. But knowing what each function does isn't enough to allow a developer to take full advantage of QuickTime. QuickTime Toolkit *fills in the gap—providing plenty of practical examples of "how" to use QuickTime to perform all kinds of useful tasks. More importantly, [this book] goes beyond "how" and into "why"—providing readers with a deeper understanding of QuickTime and how to benefit from using it in their own products.*

—Peter Hoddie
cofounder of Kinoma and former QuickTime architect

[The author] manages to present all components of the occasionally difficult QuickTime framework in a clear—even entertaining—fashion. His numerous examples and sample code snippets are clear and well thought out and are great starting points for new projects. QuickTime Toolkit *fills some gaps in Apple's official documentation and is an essential book for anyone preparing to dive into the powerful depths of low-level QuickTime programming.*

—Jurgen Schaub
founder, BOPJET Media, and QuickTime abuser

When QuickTime application developers get stuck, one of the first places they look for help is example code from the QuickTime column in MacTech. *Finally, these well-crafted examples and clear descriptions are available in book form—a must-have for anyone writing applications that import, export, display, or interact with QuickTime movies.*

—Matthew Peterson
University of California, Berkeley; the M.I.N.D.
Institute; and author of *Interactive QuickTime*

QuickTime Developer Series

Apple's QuickTime is a way to deliver multimedia—video, sound, styled text, MIDI, Flash, virtual reality, 3D models, sprites, and more—wrapped in a package that will play on Windows or Macintosh computers, on CD-ROM, over the Internet, in a browser window, a PDF document, a PowerPoint presentation, a Word document, or all by itself. The **QuickTime Developer Series**, developed in close cooperation with Apple, is devoted to exploring all of the capabilities of this powerful industry standard. Books in the series are written by experienced developers, including engineers from within the development team at Apple. All of the books feature a practical, hands-on approach and are prepared according to the latest developments in QuickTime technology.

QuickTime Toolkit, Volume One:
Basic Movie Playback and Media Types
Apple

QuickTime Toolkit, Volume Two:
Advanced Movie Playback and Media Types
Apple

Interactive QuickTime: Authoring Wired Media
Matthew Peterson

QuickTime for the Web:
For Windows and Macintosh, Third Edition
Apple

QuickTime Toolkit

Volume Two: Advanced Movie Playback and Media Types

Apple

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

ELSEVIER Morgan Kaufmann Publishers is an imprint of Elsevier

MORGAN KAUFMANN PUBLISHERS

Senior Editor	Tim Cox
Publishing Services Manager	André Cuello
Project Editor	Anne B. McGee
Project Management	Elisabeth Beller
Editorial Coordinator	Rick Camp
Cover Design	Laurie Anderson
Cover Image/Photo	© Digital Vision/Getty Images
Series Text Design	Rebecca Evans
Composition	Nancy Logan
Illustration	Dartmouth Publishing, Inc.
Copyeditor	Yonie Overton
Proofreader	Jennifer McClain
Indexer	Steve Rath
Interior Printer	The Maple-Vail Book Manufacturing Group
Cover Printer	Phoenix Color Corporation

Morgan Kaufmann Publishers is an imprint of Elsevier.
500 Sansome Street, Suite 400, San Francisco, CA 94111

Library of Congress Cataloguing-in-Publication
Application submitted.

ISBN: 0-12-088402-X

For information on all Morgan Kaufmann publications,
visit our website at *www.mkp.com.*

Printed in the United States of America

04 05 06 07 08 5 4 3 2 1

This book is printed on acid-free paper.

Contents

This Page Intentionally Left Blank

Preface

This book, *QuickTime Toolkit, Volume Two: Advanced Movie Playback and Media Types*, continues the investigation of QuickTime application programming that we began in Volume One. Here we'll consider a handful of the more advanced media types supported by QuickTime, including video effects, skins, Flash, and QuickTime VR. We'll also see how to capture movies from sound and video input sources, broadcast movies to the Internet or a LAN, play movies full screen, and load movies asynchronously. This book revisits some of the key topics covered in Volume One; in particular, we'll take an important second look at data references and see how they are connected with media sample references. We'll also see how to attach wired actions to Flash and QuickTime VR movies. This book ends by updating the Mac OS X version of our sample application QTShell to support the latest QuickTime and Carbon APIs.

Development Platforms

Once again, I place a premium on developing code that can be deployed on both Macintosh and Windows operating systems. For the most part, this can be achieved fairly easily by paying attention to a few simple rules that we encountered in Volume One (such as writing all atom data in big-endian format). In rare cases, however, this dedication to multiple platforms will require more intricate measures, and we'll bump into one of those cases in the very first chapter. To correctly handle the effects parameters dialog box on Windows, we'll need to write a callback function to handle Windows messages that apply to that dialog box. This callback function is not elaborate; indeed, it's barely a dozen lines of code. The trick, however, is knowing that we need it at all. That's why you need this book.

Another example of this dedication to cross-platform development occurs later in this book. We'll see how to write a plug-in module for the Mac version of the CodeWarrior integrated development environment that allows us to perform the task of adding Macintosh resources to a Window application (called *resource wacking*). Previously, it was necessary to perform at least this step on Windows, using the console-based tool RezWack. With this plug-in at hand, we can now create complete Windows applications on Macintosh computers.

Acknowledgements

These books grew out of a series of articles published over the last four years in *MacTech* magazine. I am indebted to the staff at *MacTech* for giving a sustained voice to QuickTime in their publication; thanks are due to Nick DeMello, Eric Gundrum, Dave Mark, and especially to Jessica Stubblefield and Neil Ticktin.

My colleagues at Apple, particularly in the QuickTime engineering group, have contributed in countless ways to my understanding of QuickTime and her foibles. A number of them have also directly influenced this book, either by reviewing portions of it or by providing sample code or sample content. I wish I could name them individually. I also wish I could thank by name those tireless colleagues in Apple's developer relations and technical publications groups with whom I have worked over the years. You guys rock!

It is a pleasure to thank the team at Morgan Kaufmann who worked so hard to bring these books to print in amazingly short order. Special thanks are due to Elisabeth Beller, Richard Camp, and Tim Cox. I'd also like to thank Yonie Overton, Jennifer McClain, and Nancy Logan for their contributions.

Finally, and not least, I should recognize Nathan and Raisa for their patience and support throughout the time I was writing these articles and books.

F/X

Using Video Effects in Movies

Introduction

The *QuickTime video effects architecture,* introduced in QuickTime 3, is an extensible system for applying video effects to images or video tracks. An effect applied to one image or track is called a *filter,* and an effect applied to two images or tracks is called a *transition.* QuickTime includes an implementation of the 133 standard transitions defined by the Society of Motion Picture and Television Engineers (SMPTE), as well as some additional effects developed by the QuickTime team. The SMPTE effects include various forms of wipe effects, iris effects, radial effects, and matrix effects. Of all of these, my personal favorite is a wipe effect called the *horizontal barn zigzag,* shown in Figure 1.1.

Figure 1.1 The horizontal barn zigzag wipe effect applied to two video tracks.

The additional QuickTime effects include transitions like a simple *explode* (where the first image is exploded outward to reveal the second image) and a *push* (where the first image is pushed aside by the second image). Figures 1.2 and 1.3 show these effects applied to two penguin images. QuickTime also includes a very nice *cross-fade* or *dissolve* transition (which produces a smooth alpha blending from the first image to the second) and a nifty *film noise* filter that makes a video track look like old, faded, dusty, and scratched film. Figure 1.4 shows a frame of a movie with the film noise effect.

Figure 1.2 The explode effect applied to two images.

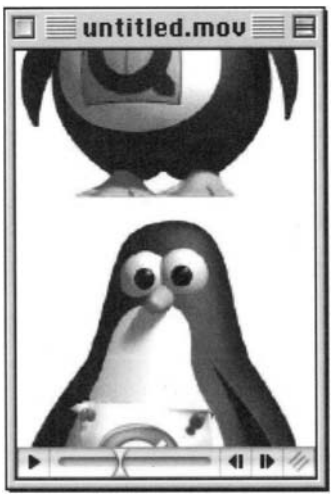

Figure 1.3 The push effect applied to two images.

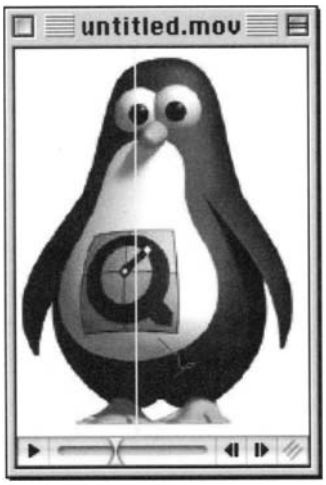

Figure 1.4 The film noise effect applied to a movie frame.

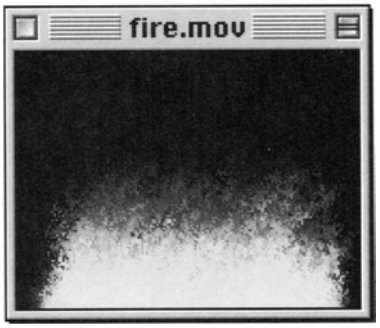

Figure 1.5 The fire effect in a movie.

There are several video effects that operate on no source images or video tracks at all, called effects *generators*. For instance, we can use the *fire* effect to generate a fairly good-looking fire (Figure 1.5), and we can use the *cloud* effect to generate a wind-pushed, moving cloud. With generators, we will usually want to composite the effect onto some other image or video track. Figure 1.6 shows the fire effect composited onto the penguin image. (Ouch, that's gotta hurt!)

The data describing an effect is stored in a video track, and the actual effect itself is generated in real time as the movie is played. These effects use extremely little data to achieve the desired visual output. For instance, a video track that specifies the fire effect is only about 60 bytes in size; when the track is played, QuickTime generates a nonrepeating, dynamic fire image.

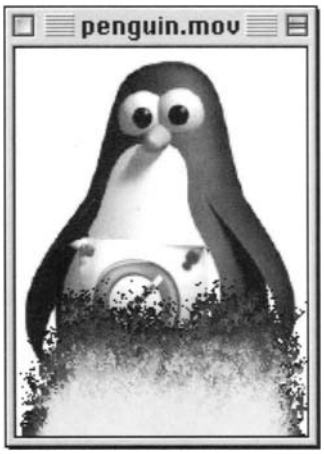

Figure 1.6 The fire effect composited onto an image.

Generators, filters, and transitions are implemented in the general QuickTime architecture as *image decompressor components* (of type `decompressorComponentType`). This means that we can reference a specific effect by providing a four-character code, which is an image decompressor component subtype. Here are a few of the available effects types:

```
enum {
  kWaterRippleCodecType                = FOUR_CHAR_CODE('ripl'),
  kFireCodecType                       = FOUR_CHAR_CODE('fire'),
  kFilmNoiseImageFilterType            = FOUR_CHAR_CODE('fmns'),
  kWipeTransitionType                  = FOUR_CHAR_CODE('smpt'),
  kIrisTransitionType                  = FOUR_CHAR_CODE('smp2'),
  kRadialTransitionType                = FOUR_CHAR_CODE('smp3'),
  kMatrixTransitionType                = FOUR_CHAR_CODE('smp4'),
  kCrossFadeTransitionType             = FOUR_CHAR_CODE('dslv'),
  kPushTransitionType                  = FOUR_CHAR_CODE('push')
};
```

This also means that we can use QuickTime video effects anywhere we might use a decompressor, not only in connection with QuickTime movies. We can just as easily apply a transition between two arbitrary images (perhaps contained in two offscreen graphics worlds). I've seen this capability used in applications that support QuickTime video effects as transitions between QuickTime VR nodes. The default behavior of QuickTime VR is simply to jump from one node to the next. It's much nicer to render some video effect, say, a nice smooth dissolve, when moving from node to node.

Figure 1.7 The Test menu of QTEffects.

In this chapter and the next, we're going to work with QuickTime video effects. We'll see how to create the fire movie shown in Figure 1.5 and how to apply a filter to a video track or image. We'll also see how to display and manage the effects parameters dialog box, which allows the user to select an effect and modify the parameters of that effect. Finally, we'll see how to apply an effect to only part of an existing movie and how to use effects as sources of sprite images.

Our sample application in these two chapters is called QTEffects; its Test menu is shown in Figure 1.7. In this chapter, we'll see how to handle all these menu items except for the fourth (which happens to be grayed out) and the final two. We'll postpone consideration of those three items to the next chapter.

QuickTime Video Effects in Movies

It's quite easy to add a video effect to a QuickTime movie. In the simplest case, where the effect lasts for the entire length of the movie, we just add an effects track to the movie. An *effects track* is a video track (of type `VideoMediaType`) whose media data is an effect description. An *effect description* is an atom container that indicates which effect to perform and which parameters, if any, to use when rendering the effect. The effect description also indicates which other tracks in the movie are to be used as the input sources for the effect. These are called the *effect source tracks* (or *effect sources*). A transition needs two source tracks; a filter needs one source track; a generator needs no source tracks. Figure 1.8 illustrates the general structure of the fire movie shown in Figure 1.5. And Figure 1.9 illustrates the general structure of a movie that contains a two-source effect (perhaps the zigzag transition shown in Figure 1.1).

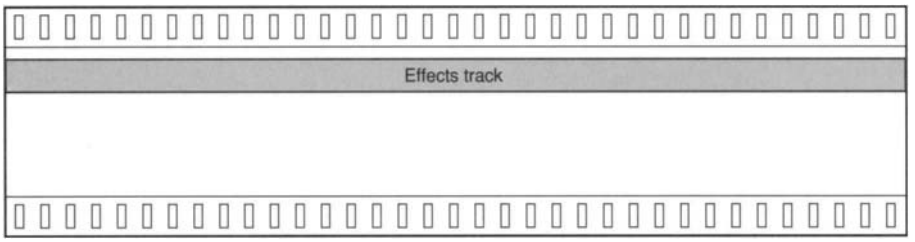

Figure 1.8 The structure of a zero-source effect movie.

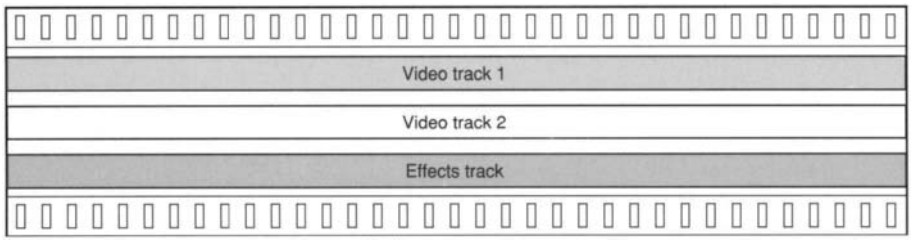

Figure 1.9 The structure of a two-source effect movie.

The source tracks for a video effect can be any tracks that have the visual media characteristic, including video tracks, sprite tracks, text tracks, and others. In particular, because an effects track is a video track, it too can be a source track for another effects track. This allows us to *stack* effects so that the output of one effect is used as input for another effect. For example, we could set up a cross-fade transition from one video track to another, and then apply a film noise filter to the resulting images. Keep in mind, however, that some effects can use a significant amount of CPU power so that stacking effects may result in movies that do not play smoothly in real time on slower machines.

As we'll see in greater detail later, we connect an effects track to its source tracks by setting up track references from the effects track to the source tracks. These references tell QuickTime where to get the data for the effects track. We also need to configure the effects track's input map so that the effects track knows how to interpret the data it receives from the source tracks. The source tracks operate as modifier tracks, whose data is not presented directly to the user; rather, their data is used as input for the effects track. This is important, particularly when we want to apply an effect to only part of a source track. You might think that we could just construct an effects track with the appropriate start time and duration, as shown in Figure 1.10. But this won't work, since once we've created a track reference from the effects track to the video track and set the effects track's input map appropriately, the video track will send *all* of its data to the effects track, not just the data in the track segment that overlaps the effects track.

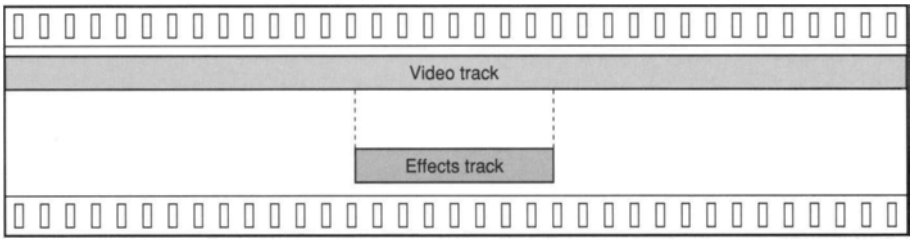

Figure 1.10 A filter applied to part of a video track (wrong).

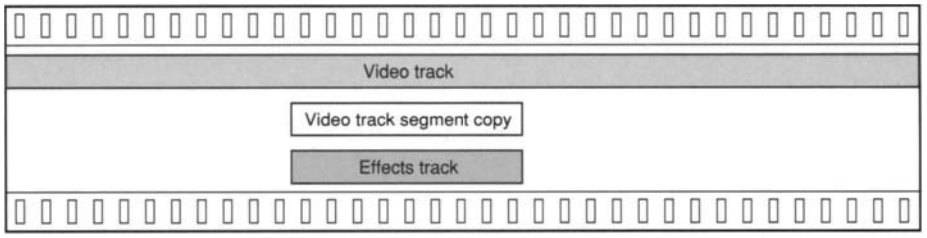

Figure 1.11 A filter applied to part of a video track.

To apply an effect to a part of a track, we can create another track that has the desired start time and duration and that references data in the video track. Then we use this new track segment as the source track for the effect, as shown in Figure 1.11. The new track segment doesn't contain a copy of the media data; instead, it contains references to the media data that already exists in the video track. So we don't increase the size of a movie file very much at all when we add effects to it. All three of the tracks shown in Figure 1.11 are enabled; to prevent the original video track from covering up the effects track, we need to make sure that the effects track has a lower track layer than the video track. We'll see exactly how to do all this in the next chapter, when we discuss applying effects to track segments.

It's worth mentioning that the QuickTime video effects architecture was originally designed to render effects in real time using software effects components (which, as we've seen, are image decompressor components). Recently, QuickTime 5 added support for hardware acceleration of effects rendering. This acceleration is used only when the user's machine has the appropriate hardware installed, and it occurs automatically (without any intervention by the effects movie creator or the playback application).

It's also worth mentioning that a video effect can have more than two sources. QuickTime 5 introduced a three-source effect, the *traveling matte* effect. In this chapter and the next, we'll always work with two or fewer sources, but our code can in fact handle up to three.

Effects Utilities

Before we begin creating effects movies, let's take a brief moment to define a couple of functions that will be useful throughout our effects code.

Creating a Sample Description

When we build an effects track, we need to pass AddMediaSample an image description that provides information about the effect. In the past, we've always created sample descriptions and image descriptions by calling NewHandleClear and then setting the fields of the structure appropriately. When we are working with effects, however, we need to use the function MakeImageDescriptionForEffect, which allocates a handle to an image description and fills in some of its fields; it also attaches an *image description extension* to the end of the image description. This extension indicates that that image description applies to an effect. For most purposes this extension is ignored, but it's necessary when we want to create stacked effects.

MakeImageDescriptionForEffect was introduced in QuickTime 4.0; if we want our code to run also under versions 3.x, we can set the USES_MAKE_IMAGE_DESC_FOR_EFFECT compiler flag to 0. Listing 1.1 shows our definition of EffectsUtils_MakeSampleDescription, which we'll call quite a few times in QTEffects to create an image description for an effect.

Listing 1.1 Creating a sample description for an effect.

```
ImageDescriptionHandle EffectsUtils_MakeSampleDescription (OSType theEffectType,
                        short theWidth, short theHeight)
{
  ImageDescriptionHandle       mySampleDesc = NULL;

#if USES_MAKE_IMAGE_DESC_FOR_EFFECT
  OSErr                        myErr = noErr;

  // create a new sample description
  myErr = MakeImageDescriptionForEffect(theEffectType, &mySampleDesc);
  if (myErr != noErr)
    return(NULL);
#else
  // create a new sample description
  mySampleDesc = (ImageDescriptionHandle)NewHandleClear(sizeof(ImageDescription));
  if (mySampleDesc == NULL)
    return(NULL);
```

```
  // fill in the fields of the sample description
  (**mySampleDesc).cType = theEffectType;
  (**mySampleDesc).idSize = sizeof(ImageDescription);
  (**mySampleDesc).hRes = 72L << 16;
  (**mySampleDesc).vRes = 72L << 16;
  (**mySampleDesc).frameCount = 1;
  (**mySampleDesc).depth = 0;
  (**mySampleDesc).clutID = -1;
#endif

  (**mySampleDesc).vendor = kAppleManufacturer;
  (**mySampleDesc).temporalQuality = codecNormalQuality;
  (**mySampleDesc).spatialQuality = codecNormalQuality;
  (**mySampleDesc).width = theWidth;
  (**mySampleDesc).height = theHeight;

  return(mySampleDesc);
}
```

Notice that we need to set a few fields of the image description even if we call `MakeImageDescriptionForEffect`.

Creating an Effect Description

It's also useful to define a utility function to build an effect description. As we've learned, an effect description is an atom container that specifies an effect and its sources. Listing 1.2 shows the definition of the utility `EffectsUtils_CreateEffectDescription`. The essential step is to add an atom of type `kParameterWhatName` and ID `kParameterWhatID` whose data is the four-character code for the desired effect.

Listing 1.2 Creating an effect description.

```
QTAtomContainer EffectsUtils_CreateEffectDescription (OSType theEffectType,
                OSType theSourceName1, OSType theSourceName2, OSType theSourceName3)
{
  QTAtomContainer      myEffectDesc = NULL;
  OSType               myType = EndianU32_NtoB(theEffectType);
  OSErr                myErr = noErr;

  // create a new, empty effect description
  myErr = QTNewAtomContainer(&myEffectDesc);
  if (myErr != noErr)
    goto bail;
```

```
  // create the effect ID atom
  myErr = QTInsertChild(myEffectDesc, kParentAtomIsContainer, kParameterWhatName,
            kParameterWhatID, 0, sizeof(myType), &myType, NULL);
  if (myErr != noErr)
    goto bail;

  // add the first source
  if (theSourceName1 != kSourceNoneName) {
    myType = EndianU32_NtoB(theSourceName1);
    myErr = QTInsertChild(myEffectDesc, kParentAtomIsContainer,
              kEffectSourceName, 1, 0, sizeof(myType), &myType, NULL);
    if (myErr != noErr)
      goto bail;
  }

  // add the second source
  if (theSourceName2 != kSourceNoneName) {
    myType = EndianU32_NtoB(theSourceName2);
    myErr = QTInsertChild(myEffectDesc, kParentAtomIsContainer,
              kEffectSourceName, 2, 0, sizeof(myType), &myType, NULL);
    if (myErr != noErr)
      goto bail;
  }

  // add the third source
  if (theSourceName3 != kSourceNoneName) {
    myType = EndianU32_NtoB(theSourceName3);
    myErr = QTInsertChild(myEffectDesc, kParentAtomIsContainer,
              kEffectSourceName, 3, 0, sizeof(myType), &myType, NULL);
  }

bail:
  return(myEffectDesc);
}
```

`EffectsUtils_CreateEffectDescription` builds an effect description with up to three *source name atoms* of type `kEffectSourceName`. The data in these atoms is a *source name* of type `OSType`. Source names are used to link the source tracks to the effects track. These names are arbitrary, but Apple recommends using names of the form '`srcX`', where *X* is an uppercase letter. In the file `EffectsUtilities.h`, we define these constants for our source names:

```
#define kSourceOneName              FOUR_CHAR_CODE('srcA')
#define kSourceTwoName              FOUR_CHAR_CODE('srcB')
#define kSourceThreeName            FOUR_CHAR_CODE('srcC')
#define kSourceNoneName             FOUR_CHAR_CODE('srcZ')
```

When we call `EffectsUtils_CreateEffectDescription`, we'll pass the constant `kSourceNoneName` for any unused sources.

Getting an Effect Type

Sometimes we might get hold of an effect description and need to know what kind of effect it describes. We can get this information by inspecting the data of the atom of type `kParameterWhatName` and ID `kParameterWhatID` that's inside that effect description. The function `EffectsUtils_GetTypeFromEffectDescription` defined in Listing 1.3 accomplishes this.

Listing 1.3 Getting the type of an effect.

```
OSErr EffectsUtils_GetTypeFromEffectDescription
        (QTAtomContainer theEffectDesc, OSType *theEffectType)
{
  QTAtom       myEffectAtom = 0;
  long         myEffectTypeSize = 0;
  Ptr          myEffectTypePtr = NULL;
  OSErr        myErr = noErr;

  if ((theEffectDesc == NULL) || (theEffectType == NULL))
    return(paramErr);

  myEffectAtom = QTFindChildByIndex(theEffectDesc, kParentAtomIsContainer,
                  kParameterWhatName, kParameterWhatID, NULL);
  if (myEffectAtom != 0) {

    myErr = QTLockContainer(theEffectDesc);
    if (myErr != noErr)
      goto bail;

    myErr = QTGetAtomDataPtr(theEffectDesc, myEffectAtom, &myEffectTypeSize,
              &myEffectTypePtr);
    if (myErr != noErr)
      goto bail;

    if (myEffectTypeSize != sizeof(OSType)) {
      myErr = paramErr;
      goto bail;
    }
```

```
    *theEffectType = *(OSType *)myEffectTypePtr;
    *theEffectType = EndianU32_BtoN(*theEffectType);

    myErr = QTUnlockContainer(theEffectDesc);
  }

bail:
  return(myErr);
}
```

Notice that we call `QTLockContainer` on the effect description, even though it isn't strictly necessary here. `QTGetAtomDataPtr` returns a pointer to the actual leaf atom data. We need to call `QTLockContainer` only when we make calls that might move memory; in this case, we're just reading a few bytes into a local variable, and this operation will not cause any memory movement. The calls to `QTLockContainer` and `QTUnlockContainer` are fairly lightweight, so we'll make them anyway.

Generators

Let's begin our hands-on work with QuickTime video effects by building a movie that uses a generator, or zero-source effect. In this case, we'll build the fire movie shown earlier in Figure 1.5. This movie has only one track, which is an effects track that has only one media sample. We'll set the dimensions of the effects track and its duration using some hard-coded values:

```
#define kDefaultTrackWidth                  160
#define kDefaultTrackHeight                 120
#define kEffectMovieDuration                (10 * kOneSecond)
```

We create the new movie file by calling `CreateMovieFile`, and then we create a new effects track and media like this:

```
myEffectTrack = NewMovieTrack(myMovie, IntToFixed(kDefaultTrackWidth),
                IntToFixed(kDefaultTrackHeight), kNoVolume);

myEffectMedia = NewTrackMedia(myEffectTrack, VideoMediaType, kOneSecond,
                NULL, 0);
```

Now we are ready to use the utility functions we defined in the previous section. We create the sample description and the effect description: