DVD Playback Services Programming Guide

Graphics & Imaging > Video

2006-03-08

Contents

Figures and Listings

Introduction to DVD Playback Services Programming Guide

This guide is written for developers who want to learn how to use DVD Playback Services, the programming interface for the DVD Playback framework. DVD Playback Services is a core technology introduced in Mac OS X version 10.3.

You can use DVD Playback Services in a Mac OS X application to display any DVD-Video recording located on an optical disc or a mass storage device such as a hard drive. The DVD Playback Services API makes it easy for applications to incorporate basic video playback features such as selecting a title from a menu and playing the title, as well as advanced features such as bookmarks. Apple's DVD Player application uses DVD Playback Services extensively.

You'll find this guide useful if you're working on:

- Applications used for authoring and publishing rich multimedia content. You can use DVD Playback Services to extend these applications to play DVD-Video content as part of a larger presentation.
- DVD-Video publishing systems. You can use DVD Playback Services for testing (or proofing) content that's under development.
- Software DVD-Video players. For example, you could use DVD Playback Services to add DVD-Video playback features to a general media player.

DVD is a big subject, and excellent technical documentation is available in bookstores and online to help you acquire a basic understanding of DVD concepts and terminology. This document assumes that you're familiar with DVD basics.

Organization of This Document

To get the most out of this guide, you should read it in this order:

- "Programming Concepts" (page 9) describes the most important programming features in DVD Playback Services.
- "Basic Programming Tasks" (page 17) shows how to implement basic DVD playback features.
- "Additional Programming Tasks" (page 27) shows how to implement some other DVD playback features, and how to debug your code.

An appendix contains a glossary that defines many of the terms used in this guide. Readers new to DVD may find it useful to read through the glossary before diving into the programming concepts.

See Also

Apple offers some additional resources to supplement the information in this guide:

- *DVD Playback Framework Reference* describes the functions, data types, and constants in DVD Playback Services.
- The sample code project *CocoaDVDPlayer* demonstrates how to write a Cocoa application that uses DVD Playback Services to play DVD-Video media.

Also explore these external (non-Apple) documents and websites:

- The book *DVD Demystified, Second Edition* (McGraw-Hill Professional, ISBN0071350268), by Jim Taylor, is the authoritative guide to DVD technology.
- The website DVD FAQ (dvddemystified.com/dvdfaq.html) contains a list of frequently asked questions (and answers) about DVD technology, maintained by Jim Taylor.
- DVD Forum (dvdforum.org) is the website of the primary industry association for the promotion of DVD technology.

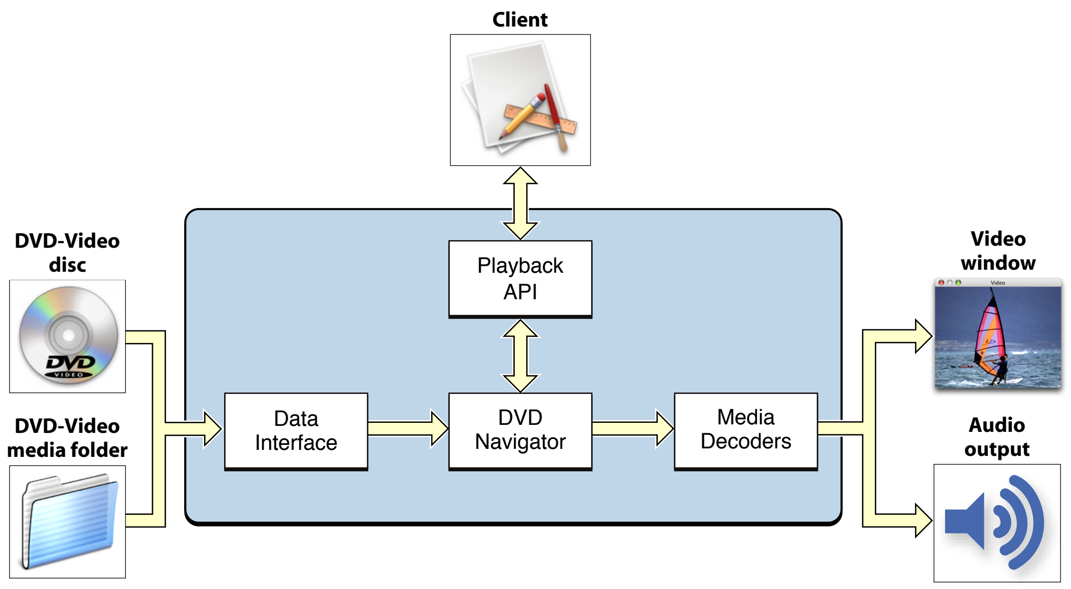
Programming Concepts

Introduced in the late 1990's, DVD-Video is a standard for storing and reproducing audio and video on DVD-ROM discs. DVD-Video is an exciting multimedia technology, and DVD Playback Services makes this technology accessible and easy to use for developers writing Mac OS X applications. DVD Playback Services makes it possible to control all aspects of DVD-Video playback without concern for the complicated details of getting the media content stream from the disc to the user. You can use DVD Playback Services to write your own DVD player, or use it to incorporate playback features into an existing application. This chapter describes the concepts you'll need to understand before you begin using DVD Playback Services.

DVD Playback Architecture

DVD Playback Services provides an abstraction layer for the programmer that hides the complexity of playback processing. The API includes a basic set of functions that are closely aligned to the controls on a typical DVD player, and other functions to implement advanced features such as bookmarks and region code updates. The diagram in Figure 1-1 shows a simplified, high-level view of how DVD Playback Services works.

Figure 1-1 DVD playback high-level architecture



The data interface reads the media and creates data streams for video, audio, subpictures, navigation, and so on. (Subpictures are bitmap overlays used for subtitles, captions, and other static visual content.) The DVD navigator manages the DVD user interface and controls playback. Menus are used for content selection, and in some cases for feature control. The content consists of up to 99 titles, and each title is divided into one or more sequential parts (chapters). Media decoders handle the decryption and decoding of compressed video and audio data streams. DVD Playback supports the MPEG-1 and MPEG-2 video codecs, and the PCM, AC-3, and MPEG audio codecs. The video channels are composited for display in a Carbon or Cocoa window. Audio data is converted into stereo analog output.

The Playback Session

DVD Playback Services uses the concept of a playback session. A playback session is a connection between a process and the DVD Playback framework. Only one session is possible at a time on a given computer system. (This is necessary because on certain computer systems the graphics I/O drivers are not re-entrant.) A new session begins when you call the `DVDInitialize` function and ends when you call the `DVDDispose` function.

Sessions are multithreaded. All drawing is handled in the main thread. If you implement a callback function and register the function with DVD Playback Services, the function is always called in a thread other than the main thread.

During a playback session, DVD Playback Services maintains state information on your behalf. You can use accessor functions such as `DVDGetTitle` to ask for specific state information, or you can receive notifications of state change events by registering a callback function. Notifications are discussed in more detail in the next section "DVD Events."

DVD Events

DVD Playback Services provides accessor functions to get the current values of various state parameters during playback. For example, you can use the `DVDGetTime` function to find out if the elapsed playback time has changed. It's possible to detect state changes by calling an accessor function repeatedly; this practice is often referred to as polling. Polling can consume alot of cycles, and its use is generally discouraged in Mac OS X applications.

To eliminate the need for polling, DVD Playback Services defines a set of notification events called DVD events. DVD events are used to notify your application of important state changes. Most DVD events include one or two integer values that indicate something about the current playback state. To receive DVD events of interest, you register one or more callback functions. You can register a single callback function to handle multiple events, or use several different callbacks.

For example, you may want to implement a clock to show the user how long the current title has been playing. The best way to keep track of the current playback time is to register a callback to receive the `kDVDEventTitleTime` event. This event passes the elapsed time and duration of the current title to your application at a regular interval that you can control.

To learn about the various types of DVD events, see "Event Codes" in the constants section of *DVD Playback Framework Reference*.

Media Folders

DVD-Video media is enclosed in a folder with a standard name. In standard-definition media, the folder is named VIDEO_TS. On a DVD disc, the media folder is located at the root level of the disc's Universal Disc Format (UDF) file system. Media folders can also be stored anywhere on a hard drive; this might occur if you're authoring the media yourself. A media folder generally contains many different files and folders in a standard configuration.

When you use DVD Playback Services to open DVD-Video media for playback, you must specify the full path to the media folder on the mounted disc volume or hard drive. For example, the path to the media on a DVD disc might be `/Volumes/MY_MOVIE/VIDEO_TS`. Before attempting to open media, DVD Playback Services checks the media folder to make sure its structure is valid. In addition, the media's region codes are checked against the region code currently assigned to the DVD drive, to make sure the media is authorized for playback in the drive.

After you open DVD-Video media, you can ask DVD Playback Services for the media's unique 64-bit identifier or media ID. You can use this media ID as a key to save information about the media in a file for later use.

Exceptions and Error Handling

Almost all the functions in DVD Playback Services return a result code, either zero (`noErr`) to indicate success or a negative integer to indicate why the function failed. DVD Playback Services defines several dozen result codes. Generally, a non-zero result code indicates one of two things:

- The operation is not permitted because of the current playback state. For example, if no media folder is open and you call the `DVDPlay` function, nothing happens and the function passes back the result code `kDVDErrorNoValidMedia`.
- The operation is not permitted by the media itself. Authors of DVD media can place constraints on various operations. When DVD Playback Services enforces one of these constraints, it returns the result code `kDVDErrorUserActionNoOp`.

More serious exceptions or errors can also occur. These errors are signaled to your application by calling one of the custom callback functions that you register. There are two types of serious errors: errors signaled with a DVD error event (`kDVDEventError`), and I/O errors signaled by calling a fatal error callback function that you implement and register. When you receive a serious error, you should end the playback session immediately.

Video Windows

A video window is simply a Carbon or Cocoa window that you create and use for DVD-Video playback. DVD Playback Services supports the use of a single window for playback. You need to specify the window's graphics port (Carbon) or window ID (Cocoa), and the bounds of the area inside the window in which you want the video to be displayed. This video area is a rectangular area somewhere inside the content area of the window; often the content area and the video area are the same rectangle.

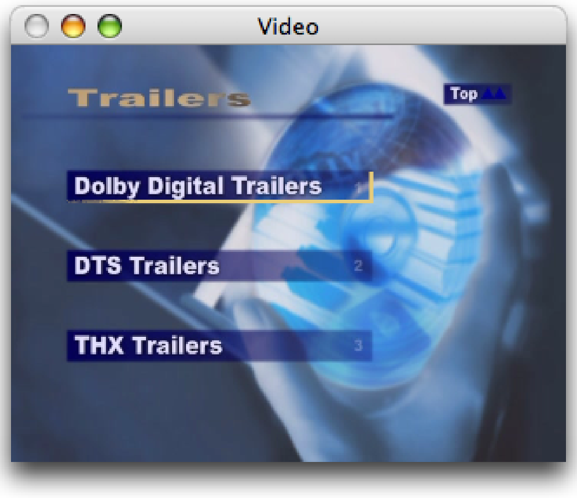
DVD Playback Services always scales the video to fit inside the video area, so it's important to specify a video area with the correct aspect ratio. Every title on a DVD is authored for one of two video aspect ratios: standard (4:3 or 1.33) or wide (16:9 or 1.78). Whenever the title changes, you are responsible for changing the aspect ratio of the video area to conform to the new aspect ratio. Some scaling is unavoidable, because the dimensions of the actual DVD-Video raster do not conform to either of these two aspect ratios. (Another way of thinking about this is that DVD pixels are rectangular, not square.) For example, the NTSC raster for standard video is actually 720 x 480 pixels. To display NTSC standard video output as it's intended to be seen, you need to rescale the video area horizontally (640 x 480), vertically (720 x 540), or both.

You also need to tell DVD Playback Services which display device it should use for the video output. For optimal performance, the window should always be positioned on a single screen. On systems with more than one screen, the user may attempt to move the video window so that it straddles two screens at once. If this occurs, your application should move the entire window onto the screen that contains the largest window area.

Menus and Menu Navigation

DVD-Video media generally contains one or more menus, including a disc or top menu. A DVD menu is an on-screen menu that presents one or more buttons and other visual information to help the user make a choice. The buttons specify titles to play, settings, or other menus. The visual information can include button highlights and animated backgrounds. The information needed to display these menus is contained in the media itself; no Carbon or Cocoa UI elements are needed. (It's important not to confuse DVD menus with application menus such as the Controls menu in DVD Player.)

Figure 1-2 A DVD menu



DVD Plackback Services provides an abstraction layer that eliminates the complexity of displaying and operating DVD menus. You simply need to notify DVD Playback Services whenever the user moves or clicks the mouse inside the video window, presses an arrow key to navigate between buttons, or presses the return or enter key to activate a button. DVD Playback Services provides a set of functions you can call whenever one of these events occurs. These functions are documented in the "Menu Navigation" section of *DVD Playback Framework Reference*.

To summarize, DVD Playback Services is responsible for displaying the DVD Menu and performing its actions. You are responsible for notifying DVD Playback Services when a user operates an input device (mouse or keyboard) to use the menu. Visual feedback and button selection in the menu is handled for you automatically. When the user selects and activates a button, DVD Playback Services performs the associated action. The action might be to play a title or scene, or branch to another menu.

Audio Output

When DVD media is playing, DVD Playback Services directs the audio output to the system's internal speakers or to an external device connected to the audio output port.

The audio output volume experienced by the user is controlled by two settings: the system volume setting and the playback volume setting. The playback volume is a subset of the system volume. DVD Playback Services allows you to set the playback volume using an integer that ranges from 0 to 255, where 255 is the current system volume. Both stereo channels use the same volume setting. DVD Playback Services also allows you to toggle the audio output on and off; you could use this feature to implement an audio mute control.

DVD Playback Services provides functions to get information about the audio stream being used in the current title, to find other audio streams, and to switch to a different audio stream. For example, you could use these functions to switch between different formats such as AC-3 and MPEG audio. These functions are documented in the "Accessing Audio Streams" section of *DVD Playback Framework Reference*.

Bookmarks

DVD Playback Services provides a way to create bookmarks that specify the media position during playback. A bookmark can represent a menu, a still picture, the elapsed time in a title, or any other location in the media. A bookmark is a block of data that can be stored in memory or written out to a file. At some later time, you can retrieve the bookmark and call the `DVDGotoBookmark` function to go to the specified position in the media and begin playing.

You can use bookmarks to implement a number of interesting features. For example, you could:

- Allow the user to create bookmarks to favorite scenes for later re-play.
- Save a bookmark to the last play position in a file, and use this bookmark to resume playback at the specified frame when the media is re-opened.

Languages

On DVDs that support multiple languages, generally the user can select a different language for the audio channel, subtitles, and menus using a language selection menu created by the DVD author. You do not need to do any special programming to take advantage of this feature.

When you open media for playback, DVD Playback Services selects a default language based on the language setting in the International Preferences panel and the languages available in the media. You can use DVD Playback Services to determine the language used in the current title or menu, and to set the default to a different language. For more information, see the section "Accessing Language Codes" in *DVD Playback Framework Reference*.

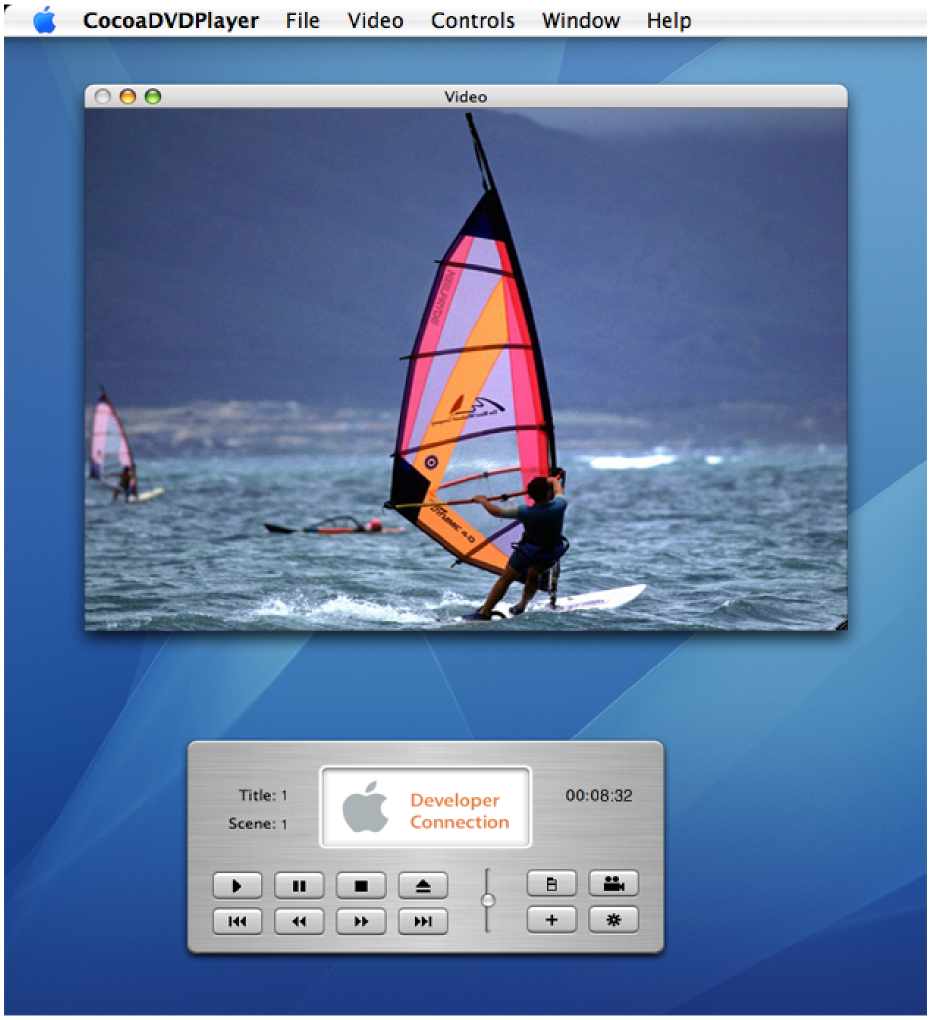
Camera Angles

The author of a DVD can create as many as nine video streams that represent the same scene viewed from different camera angles. The primary angle is designated as angle 1. You can use DVD Playback Services to find the number of different angles in a title, and to get or set the current angle. For more information, see the section "Accessing Camera Angles" in *DVD Playback Framework Reference*.

An Example DVD Player

CocoaDVDPlayer is a project in the ADC Reference Library that shows how to use DVD Playback Services to write a working DVD-Video player. The CocoaDVDPlayer user interface, shown in Figure 1-3, features a video window and a controller window that serves as the "remote control" for the player. (It's also possible to operate the player from the keyboard.)

Figure 1-3 User interface for a DVD-Video player created using DVD Playback Services



Most of the code examples in this guide are based on source code used in CocoaDVDPlayer. You may find it useful to download this project now and follow along as your read about the programming tasks in the next two chapters.

Basic Programming Tasks

In this chapter, you'll learn about the basic operations you must implement in order to play DVD-Video media. To play DVD-Video media using DVD Playback Services, you'll need to perform this sequence of operations:

1. Begin a new DVD playback session.
2. Set the window to be used for video playback.
3. Set the video display device for the window.
4. Set the bounds of the video area inside the window.
5. Open a DVD-Video media folder.
6. Play the media and respond to user actions.
7. Close the media folder.
8. End the DVD playback session.

Note: In the interest of brevity, the code examples in this guide do not always check the result codes returned from DVD Playback Services functions.

Starting a Playback Session

Initialization

When your application is ready to begin a DVD playback session, the first step is to call the `DVDInitialize` function. The result code indicates whether the API is available for use. If the result code is `noErr`, you're ready to proceed to the next step.

In this example, the application quits if DVD Playback Services cannot be initialized:

```
OSStatus err = DVDInitialize();
if (err != noErr) {
    NSLog(@"DVDInitialize returned %d", err);
    [NSApp terminate:self];
}
```

Registering an Error Handler

DVD Playback Services uses a callback mechanism to notify your application if an I/O error makes it impossible to play the media. Typically this error occurs when the optical disc is dirty or damaged. If you don't register a callback function to receive this error, your application may crash unexpectedly. Your error callback should do two things: (1) notify the user that a fatal error has occured, and (2) end the playback session (see "Ending a Playback Session" (page 25)).

This line of code shows how to register a callback function to handle a fatal error:

```
OSStatus err = DVDSetFatalErrorCallBack (MyDVDErrorHandler, (UInt32) self);
```

In this example, the callback passes the error code to a method that handles the error:

```
void MyDVDErrorHandler (DVDErrorCode inErrorCode, UInt32 inRefCon) {
    Controller *controller = (Controller *)inRefCon;
    [controller handleDVDError:inErrorCode];
}

- (void) handleDVDError:(DVDErrorCode)error {
    NSLog(@"fatal error %d", error);
    [NSApp terminate:self];
}
```

Creating a Video Window

Once the session is active, your application needs to create a window for video playback. Typically the characteristics of your video window are specified in an Interface Builder nib file, and the window is instantiated at runtime when your application loads the nib.

The recommended window type is a document window with zoom, minimize, and resize controls. The recommended initial view size is 640 x 480, which corresponds to the standard DVD aspect ratio of 4:3.

If the video window is a Cocoa window, you set the video playback window with the `DVDSetVideoWindowID` function. In CocoaDVDPlayer, the video window is a subclass of NSWindow, which has a method to get the window number. Here's the line of code that sets the window:

```
OSStatus err = DVDSetVideoWindowID ([self windowNumber]);
```

Here's the line of code that sets the video window in a Carbon application. The window is a variable of type `WindowRef`):

```
OSStatus err = DVDSetVideoPort (GetWindowPort (myWindow));
```

Choosing a Graphics Display

After the video window is created, your application needs to tell DVD Playback Services which graphics display to use for DVD playback.

Cocoa applications should set the playback display with DVDSwitchToDisplay, a convenience function that also validates the display to make sure it's supported for playback. Be sure the check the boolean value that's passed back to verify that the display is supported. If it's not supported, your application should notify the user and terminate the playback session.

A Cocoa window can find its display ID and set the display as follows:

```
CGDirectDisplayID display = (CGDirectDisplayID)
   [[[[self screen] deviceDescription] valueForKey:@"NSScreenNumber"] intValue];
Boolean isSupported;
OSStatus err = DVDSwitchToDisplay (display, &isSupported);
```

Carbon applications can find the graphics device for their video window using the Window Manager function GetWindowGreatestAreaDevice, and can set the playback display using the convenience function DVDSwitchToDevice. As with DVDSwitchToDisplay, be sure to check the boolean value that's passed back.

```
GDHandle device;
OSStatus err = GetWindowGreatestAreaDevice (myWindow, kWindowContentRgn, &device,
 NULL);
Boolean isSupported;
err = DVDSwitchToDevice (device, &isSupported);
```

If the video window is moved to a different display during playback, you should use the same procedure to set the new display.

Setting the Bounds of the Video Area

The next step is to use the DVDSetVideoBounds function to tell DVD Playback Services where to display the video content. This function takes a QuickDraw rectangle (Rect) that represents the bounds of the video area inside your window. The rectangle is expressed in window local coordinates. DVD Playback Services automatically scales the video content to fit inside this video area. Typically the dimensions of the video area are the same as the content area of the window, but this isn't a requirement.

For the video content to look right, the video area should always have one of two aspect ratios: standard (4:3) or wide (16:9). The initial aspect ratio of the video area is typically 4:3, the aspect ratio of the first title or menu on most DVDs.

Applications that use a Cocoa rectangle (NSRect) or a Quartz rectangle (CGRect to represent the bounds of the video area must convert it to a QuickDraw rectangle before calling DVDSetVideoBounds. Listing 2-1 shows how this is done for a Cocoa window in which the video area and the content view have the same bounds.

Listing 2-1 Setting the video bounds

```
- (void) setVideoBounds
{
    NSRect content = [[self contentView] bounds];                   // 1
    NSRect frame = [self frame];                                    // 2

    Rect qdRect;                                                    // 3
    qdRect.left = 0;
    qdRect.right = content.size.width;
    qdRect.bottom = frame.size.height;
    qdRect.top = frame.size.height - content.size.height;                    // 4

    OSStatus err = DVDSetVideoBounds (&qdRect);
}
```

Here's what this code does:

1. Gets the height and width of the content view.
2. Gets the height of the window, including the title bar.
3. Creates a QuickDraw rectangle to represent the video bounds.
4. Computes the position of the top edge in window local coordinates.

Whenever the user resizes the window or displays a new title with a different the aspect ratio, your application needs to call `DVDSetVideoBounds` again to notify DVD Playback Services that the video area has changed.

Opening Media

Now you're ready to open DVD-Video media for playback. Standard-definition media is stored inside a media folder named VIDEO_TS. A standard-definition DVD-Video disc has a VIDEO_TS folder in its root directory, for example. DVD Playback Services allows only one media folder to be open at a time; before you attempt to open a media folder, you should use the `DVDHasMedia` function to check to see if media is already open.

DVD Playback Services provides two functions to open media:

- To open DVD-Video media on a disc volume, you should call the `DVDOpenMediaVolume` function and pass a file object that specifies the media folder at the root level of the volume—for example, `/Volumes/CITIZEN_KANE/VIDEO_TS`.
- To open DVD-Video media on a hard drive, you should call the `DVDOpenMediaFile` function and pass a file object that specifies the full path to the media folder—for example, `/Users/Orson/Movies/CITIZEN_KANE/VIDEO_TS`.

Both functions take as input a file object of type `FSRef` that specifies the path to the media folder. Before calling one of these functions, you should validate the file object using the `DVDIsValidMediaRef` function.

Listing 2-2 shows how to use these functions:

Listing 2-2 Opening media

```
- (BOOL) openMedia:(FSRef *)media isVolume:(BOOL)isVolume
{
    Boolean isValid;
    OSStatus err = DVDIsValidMediaRef (media, &isValid);
    if (isValid)
    {
        if ([self hasMedia] == YES) {
            [self closeMedia];
        }
        if (isVolume) {
            err = DVDOpenMediaVolume (media);
        }
        else {
            err = DVDOpenMediaFile (media);
        }
    }
    return isValid;
}
```

Opening a Media Folder

DVD authoring applications such as iDVD and DVD Studio Pro can be used to create VIDEO_TS media folders on the users' hard drive. Your application may be called upon to open such a folder and play its contents. Listing 2-3 shows how a Cocoa application could display an Open dialog that allows the user to navigate to and open a media folder.

Listing 2-3 Opening a media folder

```
- (IBAction) onOpenMediaFolder:(id)sender
{
    NSOpenPanel *panel = [NSOpenPanel openPanel];                    // 1
    [panel setCanChooseFiles:NO];
    [panel setCanChooseDirectories:YES];
    [panel setAllowsMultipleSelection:NO];

    if ([panel runModalForTypes:nil] == NSOKButton)                  // 2
    {
        NSString *folderPath = [[panel filenames] objectAtIndex:0];  // 3
        const char *cPath = [folderPath                              // 4
cStringUsingEncoding:NSASCIIStringEncoding];
        FSRef fileRef;
        OSStatus err = FSPathMakeRef ((UInt8*)cPath, &fileRef, NULL);  // 5
        [self openMedia:&fileRef isVolume:NO];                       // 6
    }
}
```

Here's what this code does:

1. Creates an Open panel and configures it to open a single folder.
2. Displays the panel and checks for the OK button.
3. Retrieves a POSIX path to the selected folder.
4. Gets a C string representation of the path.
5. Converts the path string into a file object.
6. Passes the path to the `openMedia` method (see Listing 2-2 (page 20)).

Playing Media

If media is open and playback is stopped or paused, you can begin playing media by calling the `DVDPlay` function. This function does not take any arguments to control the position at which playback begins. Instead, DVD Playback Services maintains a position marker called the **last play bookmark**. This bookmark specifies the last known playback position. In a new playback session, the last play bookmark is cleared to indicate that playback should start at the beginning of the media. Your application can update or reset this bookmark using the functions `DVDSetLastPlayBookmark` and `DVDClearLastPlayBookmark`.

The initial media playback stream usually takes the viewer to an on-screen menu. To play a feature title or branch to a different menu, the user operates the mouse or keyboard to select and press a menu button. In response, DVD Playback Services uses instructions embedded in the media itself to update the last play bookmark and begin playing the specified choice. In this situation, your application does not need to use `DVDPlay`.

While media is playing, you can pause or stop playback at the current frame. If you call the `DVDStop` function twice in succession, the last play bookmark is cleared.

Responding to User Actions in DVD Menus

When a title is playing and the user decides to display a menu, you should respond by calling the `DVDGoToMenu` function. Conversely, you should call the `DVDReturnToTitle` function when a menu is on the screen and the user decides to return to the current title.

If the user clicks the Menu button in CocoaDVDPlayer, for example, this action method switches between a title and its root menu:

```
- (IBAction) onToggleMenu:(id)sender
{
    if ((mDVDState == kDVDStatePlaying) ||
        (mDVDState == kDVDStatePlayingStill) ||
        (mDVDState == kDVDStatePaused))
    {
        Boolean onMenu = false;
        DVDMenu whichMenu;
        OSStatus err = DVDIsOnMenu (&onMenu, &whichMenu);
        if (onMenu) {
            err = DVDReturnToTitle();
        } else {
            err = DVDGoToMenu (kDVDMenuRoot);
        }
    }
}
```

When a DVD menu is first displayed, generally the first (topmost) button is selected, as shown in Figure 1-2 (page 13). As the user operates the mouse or keyboard to select and press the desired button, your application responds by notifying DVD Playback Services. DVD Playback Services responds by selecting or activating the correct button on screen.

To implement this behavior, your application needs to perform the following actions:

- If the user presses an arrow key or chooses a navigation command such as Up or Down from an application menu, call the `DVDDoUserNavigation` function, passing a constant such as `kDVDUserNavigationMoveDown` to select the next button.
- If the user presses the Enter or Return key, call the `DVDDoUserNavigation` function to activate the selected button, passing the constant `kDVDUserNavigationEnter` to activate the selected button.
- If the user moves the mouse somewhere inside the video area, call the `DVDDoMenuMouseOver` function and pass in the location of the mouse.
- If the user clicks the mouse inside the video area, call the `DVDDoMenuClick` function and pass in the location of the mouse.

Listing 2-4 shows how to handle mouse events in the video window. In a Cocoa application, to receive mouse-moved events you'll also need to send the `setAcceptsMouseMovedEvents:YES` message to the window.

Listing 2-4 Handling mouse events in the video window

```
- (void) sendEvent:(NSEvent *)theEvent
{
    SInt32 index = kDVDButtonIndexNone;                              // 1

    NSPoint location = [theEvent locationInWindow];                  // 2
    Point portPt;
    portPt.h = location.x;
    portPt.v = [self frame].size.height - location.y;

    switch ([theEvent type])
    {
        OSStatus err;
        case NSMouseMoved:
            err = DVDDoMenuMouseOver (portPt, &index);               // 3
            break;
        case NSLeftMouseDown:
            err = DVDDoMenuClick (portPt, &index);                   // 4
            break;
        default:
            break;
    }

    /* sync the cursor */
    NSCursor *cursor;
    if (index != kDVDButtonIndexNone) {                              // 5
        cursor = [NSCursor pointingHandCursor];
    } else {
            cursor = [NSCursor arrowCursor];
    }
    [cursor set];

    [super sendEvent:theEvent];
}
```

Here's what the code does:

1. Declares a variable to receive the button index if the mouse is over a button.

2. Gets the mouse location in Quartz coordinates (origin in lower left corner of window).
3. Displays visual feedback in the video window when the mouse cursor is inside a DVD button.
4. Activates a DVD button if the mouse cursor is inside the button when the mouse is clicked.
5. Sets the cursor to a pointing hand when the mouse is over a button.

Listing 2-5 shows how to handle keyboard events when a menu is displayed.

Listing 2-5 Handling keyboard events

```
- (BOOL) onKeyDown: (NSEvent *)theEvent
{
    NSString *keyString = [theEvent characters];                           // 1
    unichar key = [keyString characterAtIndex:0];                          // 2
    BOOL keyIsHandled = YES;

    switch (key) {                                                         // 3
        case NSUpArrowFunctionKey:
            DVDDoUserNavigation(kDVDUserNavigationMoveUp);
            break;
        case NSDownArrowFunctionKey:
            DVDDoUserNavigation(kDVDUserNavigationMoveDown);
            break;
        case NSLeftArrowFunctionKey:
            DVDDoUserNavigation(kDVDUserNavigationMoveLeft);
            break;
        case NSRightArrowFunctionKey:
            DVDDoUserNavigation(kDVDUserNavigationMoveRight);
            break;
        case NSCarriageReturnCharacter:
        case NSEnterCharacter:
            DVDDoUserNavigation(kDVDUserNavigationEnter);
            break;
        case ' ':
            if (mDVDState == kDVDStatePlaying)
                [self onPause:self];
            else if (mDVDState == kDVDStatePaused)
                [self onPlay:self];
            break;
        default:
            keyIsHandled = NO;
            break;
    }

    return keyIsHandled;                                                   // 4
}
```

Here's what this code does:

1. Gets a string that contains a Unicode representation of one or more keys pressed by the user.
2. Gets the first Unicode character in the string.
3. Handles the key. The arrow keys are used to navigate between buttons; the enter and return keys are used to press a button; the spacebar toggles between play and pause.

4. Returns a status flag (indicating whether the event was handled) to the window that received the event.

Closing Media

When the user is finished playing a media volume or folder, you should close the media. Before attempting to close the media, you should call the `DVDHasMedia` function to make sure the media is currently open. Depending on how the media was previously opened, you need to call one of two functions to close it:

- If you opened the media with the `DVDOpenMediaFile` function, you should close it by calling the `DVDCloseMediaFile` function.
- If you opened the media with the `DVDOpenMediaVolume` function, you should close it by calling the `DVDCloseMediaVolume` function.

Ending a Playback Session

When the user is finished using DVD Playback Services, or when a fatal error occurs, you should end the playback session. There are two basic tasks: (1) unregister all DVD event callbacks, and (2) call the function `DVDDispose`, which allows another process to use DVD Playback Services.

In this example, the event callback registration number is used as a flag to make sure a playback session is active. You'll learn how this registration number is obtained in "Registering for DVD Events" (page 27).

```
- (void) endSession
{
    if (mEventCallBackID) {
        OSStatus err = DVDUnregisterEventCallBack (mEventCallBackID);
        err = DVDDispose();
        mEventCallBackID = 0;
    }
}
```

Additional Programming Tasks

In this chapter, you'll learn about other features such as DVD events and bookmarks, and how to debug your application code.

Registering for DVD Events

DVD Playback Services defines a set of DVD events to notify your application when important state changes occur. Responding to these events is optional. As a general rule, however, you should design your application to respond to events rather than poll for state changes.

To receive DVD events, you need to register one or more event callback functions. To register an event callback, you call the `DVDRegisterEventCallBack` function, passing in a pointer to the callback and a list of events the callback should receive. The function passes back a registration number that you will need to use when you unregister the callback.

Listing 3-1 shows how to register a callback to receive several events of interest.

Listing 3-1 Registering a DVD event handler

```
DVDEventCode eventCodes[] = {
    kDVDEventAngle,
    kDVDEventDisplayMode,
    kDVDEventError,
    kDVDEventPlayback,
    kDVDEventPTT,
    kDVDEventTitle,
    kDVDEventTitleTime,
    kDVDEventVideoStandard,
};                                                                    // 1

OSStatus err = DVDRegisterEventCallBack (
    MyDVDEventHandler,
    eventCodes,
    sizeof(eventCodes)/sizeof(DVDEventCode),
    (UInt32)self,
    &mEventCallBackID);                                               // 2
```

Here's what this code does:

1. Creates an array of the events that our callback will receive.
2. Registers our callback to receive these events. `DVDRegisterEventCallBack` passes back a registration number that is needed later to unregister the callback; see "Ending a Playback Session" (page 25).

DVD Playback Services invokes your DVD event callback function in a thread other than your application's main thread. Your response to a DVD event might include drawing operations, which need to execute in the main thread. To ensure that your event-handling code executes safely, your callback function can pass the event information to a method that is guaranteed to execute in the main thread. Listing 3-2 shows how to do this in a Cocoa application.

Listing 3-2 Decoupling a DVD event from the callback thread

```
void MyDVDEventHandler (
    DVDEventCode inEventCode,
    UInt32 inEventData1,
    UInt32 inEventData2,
    UInt32 inRefCon
)
{
    Controller *controller = (Controller *)inRefCon;

    DVDEvent *dvdEvent = [[DVDEvent alloc] initWithData:inEventCode
        data1:inEventData1
        data2:inEventData2];                                          // 1

    [controller performSelectorOnMainThread:@selector(handleDVDEvent:)
        withObject:dvdEvent
        waitUntilDone:FALSE];                                         // 2

    [dvdEvent release];                                               // 3
}
```

Here's what this code does:

1. Creates an object that encapsulates the DVD event information. The class for this object is defined in the CocoaDVDPlayer project.
2. Passes the object to our event-handling method, which executes asynchronously in the main thread.
3. The object is retained in our event handler, so it's safe to release it here.

Adjusting the Audio Output

The `DVDSetAudioVolume` function adjusts the audio output volume or level during playback. This function takes a single argument, an integer that can range from 0 to 255. The value 0 represents no output, and the value 255 represents the current system audio output volume. Both stereo channels use the same setting.

Many multimedia applications provide a slider control to adjust the audio output. For users who want to use the keyboard instead of the mouse to operate this control, you should also provide a way to change the audio output volume in increments. Listing 3-3 (page 28) shows how to implement a function that adjusts the audio volume using increments of 16. This function takes a single argument that represents the change direction (up or down).

Listing 3-3 Setting the DVD audio output volume

```
- (UInt16) setAudioVolume:(BOOL)up
```

```
{
    UInt16 minLevel, curLevel, maxLevel;
    OSStatus err = DVDGetAudioVolumeInfo (&minLevel, &curLevel, &maxLevel);    // 1

    UInt16 newLevel = curLevel;
    UInt16 delta = (maxLevel - minLevel + 1) / 16;                             // 2

    if (up) {                                                                  // 3
        newLevel = MIN(curLevel + delta, maxLevel);
    }
    else {
        newLevel = MAX(curLevel - delta, minLevel);
    }

    err = DVDSetAudioVolume (newLevel);                                        // 4
    return newLevel;                                                           // 5
}
```

Here's what this code does:

1. Retrieves the current audio settings. The minimum and maximum levels are 0 and 255.
2. Calculates the size of each increment. In this example, the increment is 256/16 = 16.
3. Calculates the new audio setting, clamping the result if necessary.
4. Sets the audio volume.
5. Returns the new setting; the caller uses this value to adjust the slider.

Using Bookmarks

Bookmarks are objects that specify the current media position during playback. You are responsible for allocating and managing the memory needed when you create a bookmark. Before attempting to create a bookmark, media must be open and playing.

Creating a bookmark is a three-step process:

1. Determine the size of a bookmark. You do this by calling the `DVDGetBookmark` function without supplying a pointer to any bookmark memory.
2. Use the `malloc` function to allocate memory for a new bookmark. You don't need to initialize the contents of the memory.
3. Call `DVDGetBookmark` a second time to create the bookmark, passing in a pointer to your newly-allocated memory. You are responsible for releasing the memory when you are finished using it.

To use a bookmark to return to a frame and resume playback, you should call the convenience function `DVDGotoBookmark`. Before calling this function, media must be open but not necessarily playing.

Listing 3-4 shows how to write a simple bookmark class.

Listing 3-4 A simple bookmark class

```
@interface Bookmark : NSObject {
    void * data;
    UInt32 size;
}
- (void) gotoBookmark;

@end

@implementation Bookmark

- (id) init {
    if ((self = [super init]) != nil) {
        data = NULL;
        size = 0;
        OSStatus err = DVDGetBookmark (NULL, &size);                  // 1
        data = malloc (size);                                         // 2
        err = DVDGetBookmark (data, &size);                           // 3
    }
    return self;
}

- (void) gotoBookmark {
    OSStatus err = DVDGotoBookmark (data, size) //4;
}

- (void) dealloc {
    free (data);                                                      // 5
    [super dealloc];
}

@end
```

Here's what this code does:

1. Gets the size of a bookmark.
2. Allocates memory for bookmark data.
3. Creates a bookmark to the current playback location.
4. Positions the media at this bookmark and resumes playback.
5. Frees the memory for this bookmark.

Using Media Identifiers

If you want to save a bookmark in a file for reuse the next time the user opens the media, you need to devise a way to associate the bookmark with the media folder to which it refers. DVD Playback Services can generate a media identifier that you can use as an associative key. For example, you could use the media identifier to name the file that contains the bookmarks for that media folder. To obtain the media identifier for the media folder that's currently open, you call the `DVDGetMediaUniqueID` function. This function returns an 8-byte array that you can convert into a string of hexadecimal characters.

Listing 3-5 shows how to retrieve and display a media identifier.

Listing 3-5 Displaying the media identifier

```
DVDDiscID id;
DVDGetMediaUniqueID (id);
NSLog(@"Media ID: %.2x%.2x%.2x%.2x%.2x%.2x%.2x%.2x",
    id[0], id[1], id[2], id[3], id[4], id[5], id[6], id[7]);
```

Updating the Region of a DVD Drive

A region code is an integer that identifies a marketing region for DVD-Video media. The countries of the world have been grouped into different regions to accommodate the release patterns of movies by the major movie studios. DVD-Video discs are generally assigned one or more of these region codes when they are manufactured. Every DVD drive is compatible with a single region: region 1 is the United States and Canada, for example, and region 2 is Japan and Europe.

For a given DVD drive, DVD Playback Services will not play a DVD-Video disc if none of its designated regions match the region assigned to the drive. DVD Playback Services checks for a matching region code whenever you open DVD media. The relevant functions—`DVDOpenMediaFile` and`DVDOpenMediaVolume`—indicate a region mismatch by returning the result code `kDVDErrorMismatchedRegionCode`.

If you are writing an application that uses DVD Playback Services, you can assign a new region code to a DVD drive for the purpose of enabling playback of a DVD-Video disc marketed for a particular region. This situation might arise whenever an application has a playback session open and the user inserts a disc that does not match the drive region. This feature is tricky to implement correctly, however, and Apple's DVD Player is already programmed to handle drive region changes. If your application encounters a region mismatch during playback, the recommended procedure is to close the session and direct the user to launch DVD Player.

If you decide to use DVD Playback Services to change a DVD drive's region:

- You must provide an authorization object from a user with administrator privileges. For more information, see *Authorization Services Programming Guide*.
- Be aware that region code assignments may be made a maximum of only five times per drive. This restriction is built into the hardware—after the fifth region code assignment, the drive is permanently assigned to this region and cannot be changed.
- Don't create a menu item for changing the region. It's not necessary to make users aware of regions and region codes unless there is actually a region mismatch.

Debugging

If you want to debug an application that uses DVD Playback Services, at runtime the application must link to a debug version of the shared library inside the DVD Playback framework. This section explains why this is the case and shows how to make this happen.

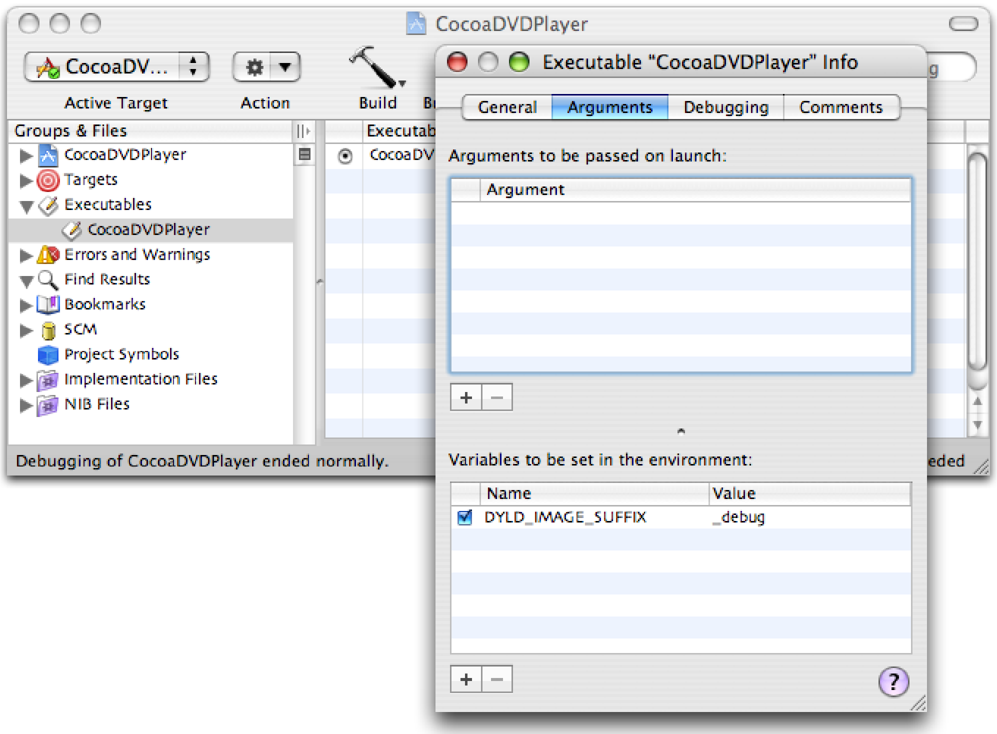
When you launch your application using the Xcode debugger or with GDB directly, you may find that the application terminates when it calls the `DVDInitialize` function to start a playback session. This is a security feature to protect the decryption code in the framework. To make debugging possible, the DVD Playback framework also contains a debug version of its shared library. The debug version does not contain the decryption code, so it plays non-encrypted media only. DVD discs are not always encrypted. For example, a DVD authored with the iDVD application is free of encryption. Another option is to use the DVD disc that's included with Jim Taylor's book *DVD Demystified*.

You need to configure your environment so that when your application launches, the dynamic loader `dyld` binds to the debug version of DVD Playback. The simplest solution is to define an environment variable that causes the dynamic loader to search for libraries with the suffix `_debug`. If you're debugging your application from a Terminal window, you can define the environment variable with this command:

```
export DYLD_IMAGE_SUFFIX=_debug
```

If you're using the Xcode debugger, Figure 3-1 shows how to use the executable inspector window to define this environment variable:

Figure 3-1 Defining an environment variable in Xcode



Name	Value
✓ DYLD_IMAGE_SUFFIX	_debug

A possible disadvantage of this solution is that it causes the dynamic loader to load debug versions of other system frameworks as well. If you're using the Xcode debugger, this may not interfere with your debugging session. If you're using GDB in a Terminal window, the additional log messages can be distracting. To learn how to load the debug version of DVD Playback without loading other debug libraries, see ADC Technical Note TN2124.

Using Result Codes

Almost all of the functions in DVD Playback Services return a result code that indicates whether the function succeeded or failed. You should write your code to receive these result codes, even if your application doesn't always check them. By receiving these codes, you can examine the status of each function call when debugging.

Glossary

angle In DVD-Video, a specific view of a scene, usually recorded from a certain camera angle. Different angles can be chosen while viewing the scene.

aspect ratio The width-to-height ratio of an image. For example, a 4:3 aspect ratio means the horizontal size is a third again wider than the vertical size. Every title on a DVD is authored for one of two aspect ratios: standard (4:3) or wide (16:9).

bookmark A data object that specifies the current media position during playback. Because the byte length of a bookmark is known, you can save the bookmark in a file for later use.

chapter 1. In DVD-Video, a division of a title. Technically called a part of title (PTT). 2. A method of organizing different scenes of a movie for easy navigation and access. DVDs are indexed by chapter, similar to the way a CD has a track. DVD players allow you to skip to a particular chapter or scene.

disc menu The main menu from which titles are selected. The disc menu is sometimes called the title menu, which more accurately refers to the menu within a title from which chapters and other features can be selected.

DVD (digital versatile disc) A high-capacity optical disc used to store everything from massive computer applications to full-length movies. A standard single-layer, single-sided DVD can store 4.7GB of data.

DVD-Video A standard for storing and reproducing audio and video on DVD-ROM discs, based on MPEG-2 video compression, Dolby Digital and MPEG audio, and other proprietary data formats.

DVD@ccess An Apple technology you can use to add interactivity to a DVD when played on a computer. DVD@ccess makes it possible to open a web browser to display HTML files, or open a program to view PDF, PICT, or JPEG files.

DVD event A notification of a state change in DVD Playback Services during playback. An event is specified with a type identifier and associated data. Client applications can register callback functions to receive events of interest.

DVD player 1. A hardware product that decodes and plays DVD-Video media stored on an optical disc. The output device is generally a television set, although some players have built-in displays and speakers. 2. A computer software program that decodes and plays DVD-Video media stored on an optical disc or a mass storage device such as a hard drive.

format standard A television video broadcast format standard. DVD Playback Services supports two format standards: the NTSC format used in North America and Japan, and the PAL format used in Europe and other continents.

media folder The video directory on a DVD disc volume. See "VIDEO_TS."

media ID A unique identifier assigned to a media folder by DVD Playback Services. This identifier can be used as a key when saving information about media playback, such as bookmarks.

play state During playback, a DVD can be in one of four play states—playing at a normal rate, paused, scanning forward, or scanning backward.

region code A code identifying one of the world regions for restricting DVD-Video playback. The different areas of the globe have been divided into eight separate regions to accommodate the varying release patterns of movies by the major studios. Therefore, each DVD player is compatible with a certain region: Region 1 for the United States and Canada, for example, and Region 2 for Japan and Europe. A DVD designated Region 0 can be played on any player regardless of its nationality.

scan direction A forward or backward direction with respect to the video stream.

scan rate A constant used in DVD Playback Services to specify the speed of play. A scan rate of 1x represents the normal playback speed; other scan rates are multiples of the normal speed.

scene See “chapter.”

stream In DVD-Video, a flow of data that contains information of a specific type, such as video, audio, subpictures, navigation data, and so on.

subpicture A graphic bitmap overlay used in DVD-Video to create subtitles, captions, karaoke lyrics, menu highlighting effects, and so on.

title The largest unit of a DVD-Video disc (other than the entire volume or side). Usually a movie, TV program, music album, or the like. A disc can hold up to 99 titles, which can be selected from the disc menu.

VIDEO_TS The file name used for the video directory or folder on a standard-definition DVD disc volume. Files inside this directory contain pointers to the sectors on the disc which hold the program streams.

Document Revision History

This table describes the changes to *DVD Playback Services Programming Guide*.

Date	Notes
2006-03-08	Fixed typographical errors.
2005-12-06	New document that describes how to use DVD Playback Services to play DVD-Video media.