Quartz Display Services Programming Topics

Graphics & Imaging > Quartz

2006-06-28

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

Simultaneously published in the United States and Canada.

Contents

Figures and Listings

Introduction to Quartz Display Services Programming Topics

Quartz Display Services is an API that provides direct access to certain low-level features in the Mac OS X window server. Quartz Display Services addresses two important types of functionality: the configuration and control of display hardware. (In these functional areas, Quartz Display Services replaces and extends the capabilites of two older APIs, the Carbon Display Manager and DrawSprocket.)

This document is a collection of short articles that provides an overview of Quartz Display Services and shows how to use this API to accomplish some basic tasks. These articles are recommended reading for software developers working on applications such as games and media players that need advanced control of displays.

Organization of This Document

This document contains the following articles:

- "Overview of Quartz Display Services" (page 9) gives a brief introduction and defines some important terms.
- "Getting Information About Displays" (page 15) briefly describes some of the accessor functions and shows how to retrieve display properties from a display mode dictionary.
- "Capturing Displays" (page 17) shows how to get exclusive use of a display for full screen drawing.
- "Changing Display Modes" (page 19) shows how to switch a display to a different display mode.
- "Configuring Displays Using a Transaction" (page 23) shows how to reconfigure one or more displays in a single operation.
- "Using Fade Effects" (page 25) shows how to fade displays during mode transitions or other configuration changes.
- "Notification of Configuration Changes" (page 29) shows how to use a notification callback to learn about display configuration changes.
- "Controlling the Mouse Cursor" (page 31) shows how to control the visibility and location of the mouse cursor.

See Also

These additional resources are available in the ADC Reference Library:

- *Quartz Display Services Reference* describes the functions, data types, and constants in Quartz Display Services.
- In *OpenGL Programming Guide for Mac OS X*, the chapter "Drawing to the Full Screen" shows how to use Quartz Display Services to switch to full-screen display mode and change screen resolutions.

- In *Quartz Composer Programming Guide*, the chapter "Using QCRenderer to Play a Composition" shows how to use Quartz Display Services and Quartz Composer to render a composition to the full screen.

Overview of Quartz Display Services

Quartz Display Services is a set of system software functions that support dynamic changes to the arrangement and display modes of the displays attached to a user's computer, as well as other display-related operations. (In this document, the term **display** refers to a graphics hardware system consisting of a frame buffer, a gamma correction table or color palette, and possibly an attached monitor.)

For example, Mac OS X applications can use Quartz Display Services to:

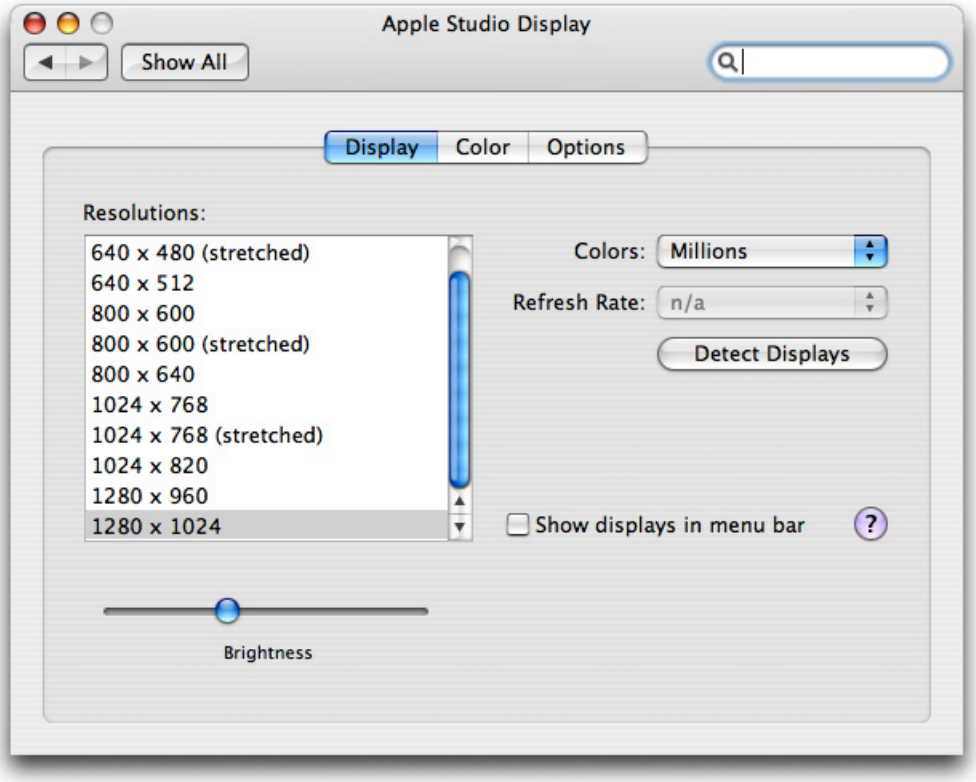
- Examine and change display modes
- Configure a set of displays in a single operation
- Capture one or more displays for exclusive use
- Perform fade effects
- Activate display mirroring
- Configure gamma color correction tables and color palettes
- Receive notification of screen update operations

Unlike the Carbon Display Manager, Quartz Display Services is a low-level API. User interface elements such as windows are not automatically repositioned when monitors are detached or display modes change. Instead, Quartz provides a notification mechanism for display state changes. High-level application frameworks such as Carbon and Cocoa can detect these state changes and make their own adjustments to the size, position, and layout of windows on the affected displays.

Mac OS X System Preferences uses Quartz Display Services to perform some of the actions in the Displays preferences pane. For example, the Display pane shown in Figure 1 (page 10) contains controls that allow the user to switch display modes. The user can:

- Choose a different display resolution
- Change the bits per pixel of a display, which determines how many colors can be displayed
- Set the refresh rate of a CRT monitor

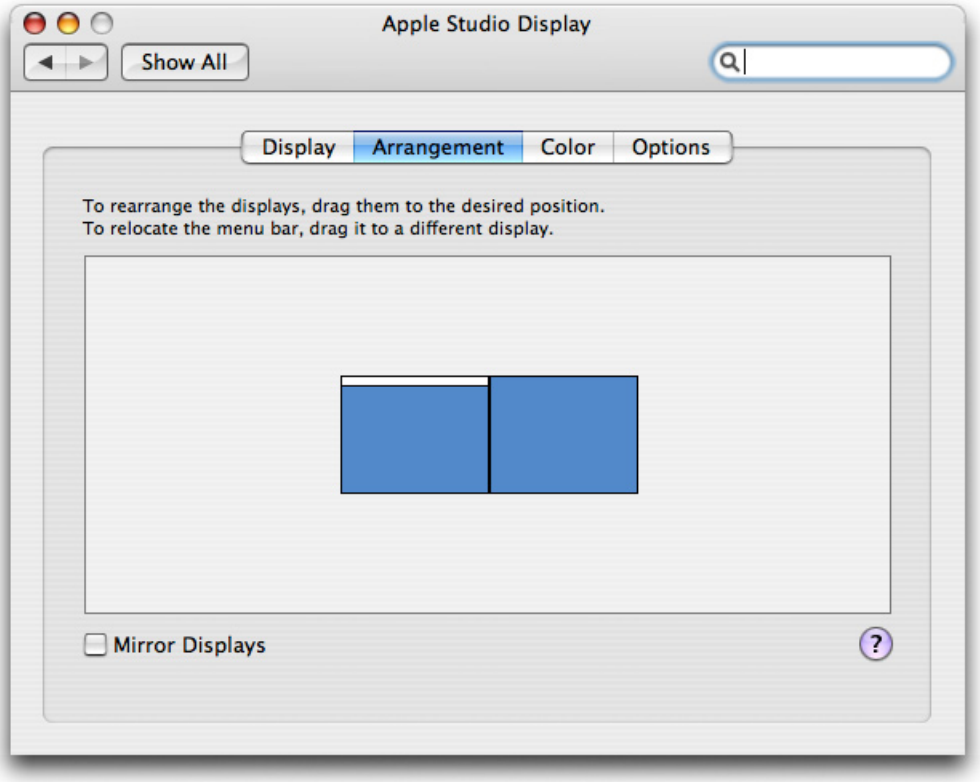
Figure 1 A Display pane



On a system with two or more attached monitors, the Arrangement pane shown in Figure 2 (page 11) contains controls that allow the user to:

- Rearrange the displays on the extended desktop
- Change the main display by moving the menu bar from one display to another
- Create a display mirroring set

Figure 2 An Arrangement pane



The next few sections introduce some basic features of displays, along with some of the more important functions in the Quartz Display Services API. The other articles in this document contain examples that show how to use this API to perform some common operations.

Display States

A display is considered to be online when the frame buffer hardware is connected to a monitor. If no monitor is attached to the frame buffer, a display is characterized as offline.

When a display is online, the display is mapped into the global display (desktop) coordinate system. The upper-left corner of a display is called the origin. The origin of a display is always specified in global display (desktop) coordinates.

The display with the origin at (0,0) is called the main or primary display. In a system without display mirroring, the display with the menu bar is typically the main display. The user can change the main display by dragging the menu bar to a different display in Displays preferences.

An online display can be active, mirroring, or sleeping. These terms are defined as follows:

- A display is active if it is awake and available for drawing.
- A display is mirroring another display if the same content is drawn to both displays simultaneously. You cannot draw on the mirroring display.

- A display is sleeping when its frame buffer and the attached monitor are in reduced power mode. A sleeping display is still considered to be a part of global display space, but you cannot draw on it.

Display IDs

When a monitor is attached and a display is online, Quartz assigns a unique display identifier (ID) of type `CGDirectDisplayID`. A display ID can persist across processes and system reboot, and typically remains constant as long as certain display parameters do not change. You can obtain an array of display IDs that correspond to all online displays in the system with the function `CGGetOnlineDisplayList`.

Typically you're more interested in active displays because they're available for drawing. You can obtain an array of display IDs that correspond to all active displays in the system with the function `CGGetActiveDisplayList`. The first display in the list is always the main display. The main display is also represented by the constant `kCGDirectMainDisplay`, which is defined as a call to the function `CGMainDisplayID`.

These functions also obtain an array of display IDs:

- `CGGetDisplaysWithPoint` obtains the display IDs for online displays whose bounds include a specified point.
- `CGGetDisplaysWithRect` obtains the display IDs for online displays whose bounds include a specified rectangle.
- `CGGetDisplaysWithOpenGLDisplayMask` obtains the display IDs for online displays that correspond to the bits set in an OpenGL display mask.

Display Modes

Every display has a set of supported modes of operation. A display mode is a set of standard properties such as resolution (width and height in pixels), bits per pixel, and refresh rate, and optional properties such as pixel stretching to fill the screen.

Each display mode is represented by a display mode dictionary. To find out what modes a display supports, you use the function `CGDisplayAvailableModes`, which returns a list of mode dictionaries. To find out the current display mode for an online display, you use the function `CGDisplayCurrentMode`, which returns a single mode dictionary. A display mode dictionary contains a set of key-value pairs that you can query using Core Foundation CFDictionary functions.

To find the optimal mode for a selected set of properties, you can use the function `CGDisplayBestModeForParameters`. For example, if you request a supported mode for a display with a resolution of 750 x 550 pixels and 24 bits per pixel, this function may return a supported mode with resolution of 800 x 600 pixels and 32 bits per pixel.

Mode dictionaries are read-only. If you want to change a specific display property such as resolution, you need to find the appropriate mode dictionary and use it to change the mode of the display. You can use `CGDisplaySwitchToMode`, a convenience function for changing the mode of a single display. For more information, see "Changing Display Modes" (page 19).

Display Arrangement

On systems with multiple displays, your application can control the arrangement of the displays in the global display space. This is done by setting the origin of each display with the function `CGConfigureDisplayOrigin`. The new origins are placed as close as possible to the requested locations, without overlapping or leaving a gap between displays. You can also use this function to designate a display as the main display by setting its origin to (0,0). For more information about using this function, see "Configuring Displays Using a Transaction" (page 23).

When your application terminates, the display arrangement returns to the current settings in Displays preferences.

Display Mirroring

On systems with multiple displays, your application can draw the same content to two or more displays simultaneously. This is called mirroring, and the displays are said to be in a mirroring set. You can create or change the configuration of a mirroring set with the function `CGConfigureDisplayMirrorOfDisplay`. One display is designated the main or primary display in the mirroring set, and all drawing is directed to this display. For more information about using this function, see "Configuring Displays Using a Transaction" (page 23).

Display mirroring and display matte generation are implemented either in hardware (preferred) or software, at the discretion of the device driver. In hardware mirroring, the graphics hardware renders the contents of a single frame buffer in two or more displays simultaneously. In software mirroring, identical content is drawn into the frame buffer of each display in the mirroring set. The display with the highest resolution and deepest pixel depth typically becomes the main display.

Quartz makes sure that all window-based content is placed on all displays in a mirroring set. Applications that draw in windows need not be concerned about supporting mirroring. Applications drawing directly to the display may need to implement a traditional device loop to properly support mirroring.

Getting Information About Displays

Quartz Display Services provides a number of functions to retrieve information about display state and display modes. This article briefly covers a few of these functions. For a complete list, see *Quartz Display Services Reference*.

Getting Display State Information

Quartz Display Services includes several accessor functions that report current properties of the display hardware, properties that are also found in the current display mode dictionary. Because these functions query the hardware directly rather than consulting the current display mode, they provide the most accurate information available about the display (display mode properties are subject to change by the device driver). These functions, listed next, are straightforward to use:

Function	Description
`CGDisplayPixelsWide`	Returns the drawable width of a display in pixel units.
`CGDisplayPixelsHigh`	Returns the drawable height of a display in pixel units.
`CGDisplayBitsPerPixel`	Returns the number of bits used to represent a pixel in the frame buffer.
`CGDisplayBitsPerSample`	Returns the number of bits used to represent a pixel component in the frame buffer.
`CGDisplaySamplesPerPixel`	Returns the number of components used to represent a pixel in the frame buffer.
`CGDisplayBytesPerRow`	Returns the number of bytes per row in the frame buffer.

The API includes other accessor functions that report additional information about the current configuration and state of the display. The following table lists a representative sample of these functions:

Function	Description
`CGDisplayIsActive`	Returns a Boolean value indicating whether a display is active or drawable).
`CGDisplayIsBuiltin`	Returns a Boolean value indicating whether a display is built-in, such as the internal display in portable systems.
`CGDisplayIsMain`	Returns a Boolean value indicating whether a display is the main display.
`CGDisplayScreenSize`	Returns the width and height of a display in millimeters.

Function	Description
`CGDisplayUsesOpenGLAcceleration`	Returns a Boolean value indicating whether Quartz is using OpenGL-based window acceleration (Quartz Extreme) to render in a display.

Getting Display Mode Information

Each online display has a current display mode and a set of available display modes. The device driver uses information in the current display mode to configure the display.

You can retrieve the values of the properties in a display mode dictionary with the `CFDictionaryGetValue` function. Then, you may need to cast the returned value to the appropriate data type. Listing 1 shows how to retrieve several mode properties and cast them to a base data type.

Listing 1 Getting display mode properties

```
CFNumberRef number;
CFBoolean booleanValue;
Boolean gui;
long mode, refresh, ioflags;

CFDictionaryRef currentMode = CGDisplayCurrentMode (kCGDirectMainDisplay);

number = CFDictionaryGetValue (currentMode, kCGDisplayMode);
CFNumberGetValue (number, kCFNumberLongType, &mode);

number = CFDictionaryGetValue (currentMode, kCGDisplayRefreshRate);
CFNumberGetValue (number, kCFNumberLongType, &refresh);

booleanValue = CFDictionaryGetValue (currentMode,
kCGDisplayModeUsableForDesktopGUI);
gui = CFBooleanGetValue (booleanValue);

number = CFDictionaryGetValue (currentMode, kCGDisplayIOFlags);
CFNumberGetValue (number, kCFNumberLongType, &ioflags);
```

Capturing Displays

If you're writing an immersive application such as a game or a presentation program, you may want to do full screen drawing.

A common approach is to capture the display you want to use. When you capture a display, you have exclusive use of the display. Other applications and system services are not allowed to use the display or change its configuration. In addition, they are not notified of display changes, thus preventing them from repositioning their windows and the Finder from repositioning desktop icons.

To capture a single display, call the function `CGDisplayCapture`. To capture all online displays at once, call `CGCaptureAllDisplays`. By default, a captured screen is filled with black color; you have the option of disabling this feature if you capture using the functions `CGDisplayCaptureWithOptions` or `CGCaptureAllDisplaysWithOptions`.

After capturing a display, there are several drawing options:

- If you're writing an OpenGL application, you can create an OpenGL full screen drawing context. For more information, see *OpenGL Programming Guide for Mac OS X*.
- You can draw directly to the screen using Quartz 2D. Use the function `CGDisplayGetDrawingContext` to obtain a full-featured graphics context for the display. The graphics context remains valid until the display is released or its configuration changes. The context's origin is the lower-left corner of the display.
- You can draw directly to the screen using your own drawing engine. Call `CGDisplayBaseAddress` or `CGDisplayAddressForPosition` to get an address in the frame buffer to which to draw.

Note: None of these drawing options require a window. Attempting to position a window over a captured display and draw into the window may be unsuccessful—or may present undesirable results such as illegible or invisible content—because of interactions between full-screen graphics and the graphics hardware.

When you are finished using a captured display, you should release it by calling `CGDisplayRelease` or `CGReleaseAllDisplays`.

Listing 1 shows how to capture the main display and draw a text string using Quartz 2D. A detailed explanation for each numbered line of code appears following the listing.

Listing 1 Capturing the main display

```
char *text = "Hello, World!";
CGDirectDisplayID display = kCGDirectMainDisplay;                          // 1
CGError err = CGDisplayCapture (display);                                  // 2
if (err == kCGErrorSuccess)
{
    CGContextRef ctx = CGDisplayGetDrawingContext (display);               // 3
    if (ctx != NULL)
    {
        CGContextSelectFont (ctx, "Times-Roman", 48, kCGEncodingMacRoman);
```

```
        CGContextSetTextDrawingMode (ctx, kCGTextFillStroke);
        CGContextSetRGBFillColor (ctx, 1, 1, 1, 0.75);
        CGContextSetRGBStrokeColor (ctx, 1, 1, 1, 0.75);
        CGContextShowTextAtPoint (ctx, 40, 40, text, strlen(text));          // 4
        sleep (4);                                                           // 5
    }
    CGDisplayRelease (display);                                              // 6
}
```

Here's what the code does:

1. Gets the display ID of the main display.
2. Captures the main display and changes the color to black. An error is returned only if the display has been captured by another application.
3. Gets a Quartz graphics context associated with the captured display.
4. Draws the text string in the lower-left corner of the screen.
5. Suspends processing for a few seconds to allow the user to read the text.
6. Releases the captured display.

Note: It's not necessary to capture a display to do full-screen drawing. Another approach is to create and draw into a borderless window the size of the display. This approach allows you to use all the features of the windowing system, it plays well with the rest of the operating system, and it reduces the complexity of display handling (for example, you don't have to worry about mirrored displays). On systems with modern graphics hardware, drawing performance in a full-screen window is almost as fast as a full-screen drawing context.

Changing Display Modes

A typical display configuration task is to change the mode of a single display. You can use the convenience function `CGDisplaySwitchToMode` to do this in a single step. This section shows two ways to find and switch to the best mode for a given display.

Setting the Mode of the Main Display

In the simple case, you want to set the mode of the main display to a mode that most closely matches your required bit depth and resolution. When you are finished drawing, you restore the previous display mode. Listing 1 shows how to do this. A detailed explanation for each numbered line of code appears following the listing.

Listing 1 Setting the mode of the main display

```
size_t desiredBitDepth = 16;
size_t desiredWidth = 1024;
size_t desiredHeight = 768;
boolean_t exactMatch;

CFDictionaryRef mode = CGDisplayBestModeForParameters(                    // 1
    kCGDirectMainDisplay,
    desiredBitDepth, desiredWidth,
    desiredHeight, &exactMatch);

if (mode != NULL) {
    /* if it is important to have an exact match, check exactMatch here */
    MyDrawToDisplayWithMode (kCGDirectMainDisplay, mode);
}

void MyDrawToDisplayWithMode (CGDirectDisplayID display, CFDictionaryRef mode)
{
    CFDictionaryRef originalMode = CGDisplayCurrentMode (display);        // 2
    CGDisplayHideCursor (display);
    CGDisplaySwitchToMode (display, mode);                                // 3
    CGDisplayCapture (display);                                           // 4

    /* full screen drawing/game loop here */

    CGDisplaySwitchToMode (display, originalMode);                        // 5
    CGDisplayRelease (display);                                           // 6
    CGDisplayShowCursor (display);
}
```

Here's what the code does:

1. Finds the best match among the available modes for the specified display.

2. Saves the current display mode.
3. Reconfigures the display to use the new display mode.
4. Captures the display to prepare for full screen drawing.
5. Restores the previous display mode.
6. Releases the captured display.

Finding the Best Mode from the Available Modes

In a more complex case, you need more control over which display you use or want to determine for yourself what "best mode" means. Listing 2 shows how to get an array of active displays, iterate over that list examining the modes that each display supports, and choose the most appropriate display and mode combination for your application.

Listing 2 Examining the available modes

```
#define MAX_DISPLAYS 32

CGDirectDisplayID displays[MAX_DISPLAYS];
CGDisplayCount numDisplays;
CGDisplayCount i;

CGGetActiveDisplayList (MAX_DISPLAYS, displays, &numDisplays);                  // 1

for (i = 0; i < numDisplays; i++)                                               // 2
{
    CFDictionaryRef mode;
    CFIndex index, count;
    CFArrayRef modeList;

    modeList = CGDisplayAvailableModes (displays[i]);                           // 3
    count = CFArrayGetCount (modeList);

    for (index = 0; index < count; index++)                                     // 4
    {
        mode = CFArrayGetValueAtIndex (modeList, index);
        if (MyBestMode (mode)) {
            MyDrawToDisplayWithMode (displays[i], mode);                        // 5
        }
    }
}

boolean_t MyBestMode (CFDictionaryRef mode)                                     // 6
{
    CFNumberRef value;
    long bitsPerPixel = 0, width = 0;

    value = CFDictionaryGetValue (mode, kCGDisplayBitsPerPixel);
    CFNumberGetValue (value, kCFNumberLongType, &bitsPerPixel);
    value = CFDictionaryGetValue (mode, kCGDisplayWidth);
```

```
    CFNumberGetValue (value, kCFNumberLongType, &width);

    if (bitsPerPixel == 32 && width == 1024)
        return true;
    else
        return false;
}
```

Here's what the code does:

1. Gets the array of active displays, which are the ones available for drawing.
2. Iterates over the array of active displays. Note that the array is zero based.
3. Gets the array of available modes for this display.
4. Iterates over the available modes for a display, calling a custom function to determine if a mode has the desired properties.
5. Calls the drawing function used in the previous example.
6. Checks two properties in the mode dictionary and returns true if the mode has the desired properties.

Configuring Displays Using a Transaction

Quartz Display Services makes it possible to configure a set of displays in a single transaction. During the execution of configuration changes, Quartz performs a standard fade effect on all online displays. The displays fade to a monochromatic color, the configuration takes place, and the displays return to normal. For more information about fade effects, see "Using Fade Effects" (page 25).

You begin a new transaction by calling `CGBeginDisplayConfiguration`. The next step is to declare what changes you want to make. For example, you can use these functions:

- `CGConfigureDisplayMode` sets the display mode of a display.
- `CGConfigureDisplayMirrorOfDisplay` adds a display to a mirroring set.
- `CGConfigureDisplayOrigin` sets a display's origin in global display space.
- `CGConfigureDisplayFadeEffect` customizes the fade effect. (Calling this function modifies the fade behavior for a single display configuration and has no permanent effect.)

After you're finished preparing the transaction, you call `CGCompleteDisplayConfiguration` to execute it. In this call you also specify the scope of the configuration change. Typically, you'll specify `kCGConfigureForAppOnly` to apply the changes for the lifetime of your application.

Listing 1 shows how to use a configuration transaction with a custom fade effect to change the display mode of a single display. A detailed explanation for each numbered line of code appears following the listing.

Listing 1 A simple configuration transaction

```
void MyDisplaySwitchToMode (CGDirectDisplayID display, CFDictionaryRef mode)
{
    CGDisplayConfigRef config;                                          // 1
    CGBeginDisplayConfiguration (&config);                              // 2
    CGConfigureDisplayMode (config, display, mode);                     // 3

    CGConfigureDisplayFadeEffect (                                      // 4
        config,
        0.6,    // fade out interval in seconds
        1.0,    // fade in interval
        0.5,    // red
        0.5,    // green
        0.5     // blue
    );

    CGCompleteDisplayConfiguration (config, kCGConfigureForAppOnly);    // 5
}
```

Here's what the code does:

1. Declares a display configuration object, a variable that provides a context for a set of display configuration changes.

2. Begins a new configuration transaction, and passes back a display configuration object.

3. Declares the display mode change for this configuration.

4. Customizes the default fade effect for this configuration. The new fade color is gray.

5. Applies the new configuration with application scope. On return, the configuration object is no longer valid.

Using Fade Effects

A display fade effect is a smooth transition to or from a monochromatic color blended with the contents of the display, often used when entering full screen mode or switching display modes. The transitions to and from monochromatic color are sometimes called fade-out and fade-in effects.

Quartz automatically fades all online displays for you during configuration changes. The time settings for the default fade effect are 0.3 seconds to fade out and 0.5 seconds to fade in. The fade color is French Blue for a normal desktop, and black for a captured display. During a configuration transaction, you can customize this built-in fade effect—see the example in "Configuring Displays Using a Transaction" (page 23).

You may want to use a custom fade effect to indicate a transition other than a configuration change. For example, you could fade out to black, capture the display, draw on the screen, and fade in from black. The next sections show two ways to accomplish this task.

Fading all Displays

Quartz Display Services provides functions that allow you to perform a custom fade effect on all online displays simultaneously. The first step is to call `CGAcquireDisplayFadeReservation` to reserve the fade engine for a specified interval. This function passes back a token that represents a new fade reservation. Your application uses this token as an argument in subsequent calls to `CGDisplayFade`. During the fade reservation interval, your application has exclusive rights to use the fade engine. To release the reservation, you call `CGReleaseDisplayFadeReservation`.

Listing 1 shows how to reserve the fade engine, perform a fade effect to and from black, and release the reservation. A detailed explanation for each numbered line of code appears following the listing.

Listing 1 Fading all displays

```
CGDisplayFadeReservationToken token;
CGDisplayErr err;

err = CGAcquireDisplayFadeReservation (kCGMaxDisplayReservationInterval, &token);      // 1
if (err == kCGErrorSuccess)
{
    err = CGDisplayFade (token, 0.3, kCGDisplayBlendNormal,
        kCGDisplayBlendSolidColor, 0, 0, 0, true);                                     // 2
    err = CGDisplayCapture (kCGDirectMainDisplay);                                     // 3
    /* draw something on the captured display */
    err = CGDisplayFade (token, 0.5, kCGDisplayBlendSolidColor,
        kCGDisplayBlendNormal, 0, 0, 0, true);                                         // 4
    err = CGReleaseDisplayFadeReservation (token);                                     // 5
}
```

Here's what the code does:

1. Reserves the fade engine for the maximum permitted interval. Your application must perform this step before it can fade the displays. During this time, your application has exclusive rights to use the fade engine.
2. Fades displays to black with a duration of 0.3 seconds. The function call is synchronous.
3. Captures the main display. The display is already black, so the user sees no change.
4. Fades displays from black to normal with a duration of 0.5 seconds. The function call is synchronous.
5. Releases the fade reservation and invalidates the token.

Fading a Single Display

To fade a single display on a system with two or more displays, you can use another approach that involves incrementally adjusting the display's gamma formula.

When you adjust the gamma values, you can't assume that the maximum gamma value is 1.0; the user might have specified a different maximum value in System Preferences. You need to retrieve the current settings and scale them appropriately.

Listing 2 shows how to fade a specified display to black and back. A loop with a fixed delay is used to obtain a smooth fade (another approach is to use a timer to ensure a fixed fade duration on different systems).

Listing 2 Fading a single display

```
const double kMyFadeTime = 1.0; /* fade time in seconds */
const int kMyFadeSteps = 100;
const double kMyFadeInterval = (kMyFadeTime / (double) kMyFadeSteps);
const useconds_t kMySleepTime = (1000000 * kMyFadeInterval); /* delay in
microseconds */

int step;
double fade;
CGGammaValue redMin, redMax, redGamma,
             greenMin, greenMax, greenGamma,
             blueMin, blueMax, blueGamma;
CGDisplayErr err;

err = CGGetDisplayTransferByFormula (display,                                  // 1
    &redMin, &redMax, &redGamma,
    &greenMin, &greenMax, &greenGamma,
    &blueMin, &blueMax, &blueGamma);

for (step = 0; step < kMyFadeSteps; ++step) {                                  // 2
    fade = 1.0 - (step * kMyFadeInterval);
    err = CGSetDisplayTransferByFormula (display,
        redMin, fade*redMax, redGamma,
        greenMin, fade*greenMax, greenGamma,
        blueMin, fade*blueMax, blueGamma);
    usleep (kMySleepTime);                                                     // 3
}
```

```
err = CGDisplayCapture (display);
/* draw something on the captured display */

for (step = 0; step < kMyFadeSteps; ++step) {                                   // 4
    fade = (step * kMyFadeInterval);
    err = CGSetDisplayTransferByFormula (display,
        redMin, fade*redMax, redGamma,
        greenMin, fade*greenMax, greenGamma,
        blueMin, fade*blueMax, blueGamma);
    usleep (kMySleepTime);                                                      // 5
}

CGDisplayRestoreColorSyncSettings();                                            // 6
```

Here's what the code does:

1. Gets the current coefficients of the gamma transfer formula for a display as the starting gamma values.
2. Fades from the current gamma by setting the color gamma function for the display, specified as the coefficients of the gamma transfer formula. Starts with the current gamma (multiplying by a factor of 1.0) and ends with black (multiplying by a factor of 0.0).
3. Suspends processing for a short interval. To get a smooth fade out effect, you either need to use a timer or insert a short delay because the call to change the display gamma returns within 100 microseconds or so, and the actual gamma is applied asynchronously during the next vertical blanking period. Without the delay, you'll get what appears as an instantaneous switch to black.
4. Fade from black (multiplying by a factor of 0.0) back to original gamma (multiplying by a factor of 1.0).
5. Suspends processing for a short interval to achieve a smooth fade in effect.
6. Restores the gamma tables to the values in the user's ColorSync display profile.

Notification of Configuration Changes

Quartz Display Services provides a general notification mechanism for applications that need to know about display configuration changes. Any application can register a display reconfiguration callback function. At several points during reconfiguration, Quartz passes to your callback function the display ID, status flags, and optional private data. During a display mode change, for example, you could use a callback to print a log message that describes the new mode.

Quartz invokes your callback function when:

- Your application calls a function to reconfigure a local display
- Your application is listening for events in the event-processing thread, and another application calls a function to reconfigure a local display
- The user changes the display hardware configuration—for example, by disconnecting a display or changing a system preferences setting

Before display reconfiguration, Quartz invokes your callback function once for each online display to indicate a pending configuration change. The `kCGDisplayBeginConfigurationFlag` flag is always set. After display reconfiguration, Quartz invokes your callback function once for each added, removed, and online display. At this time, all display state reported by Quartz Display Services, QuickDraw, and the Carbon Display Manager will be up to date. The flags indicate how the display configuration has changed.

Carbon and Cocoa already use this notification mechanism to respond to display reconfigurations. As a result, when a user or application changes a display mode, turns on mirroring, or disconnects a display, Carbon or Cocoa applications don't need to be concerned with repositioning or resizing their windows. The application frameworks handle this task automatically.

If you want to receive notifications of configuration changes, here is a brief description of the steps:

1. Register your notification callback function:

```
CGDisplayRegisterReconfigurationCallback (MyDisplayReconfigurationCallBack,
&userInfo);
```

2. When your function is called, check the parameters to see if action is required. For example:

```
void MyDisplayReconfigurationCallBack (
   CGDirectDisplayID display,
   CGDisplayChangeSummaryFlags flags,
   void *userInfo)
{
    if (flags & kCGDisplaySetModeFlag) {
        /* handle mode change for this display */
    }
}
```

3. When you no longer require notification, remove the callback registration:

```
CGDisplayRemoveReconfigurationCallback (MyDisplayReconfigurationCallBack,
&userInfo);
```

Controlling the Mouse Cursor

Quartz Display Services includes functions to control the mouse cursor. For example, when your application is in full-screen mode you may want to hide the mouse cursor, move the cursor to a different location, or disassociate mouse movement from the cursor position. This article briefly describes the functions that provide these services. To use these functions, your application must be in the foreground.

To hide or show the mouse cursor on any display, use the functions `CGDisplayHideCursor` and `CGDisplayShowCursor`. These functions take a display ID as a parameter, but the display ID is not used. Calls to these functions need to be balanced; Quartz maintains a hide cursor count that must be zero in order to show the cursor.

Quartz provides a convenient function for disassociating mouse movement from cursor position while an application is in the foreground. By passing `false` to the function `CGAssociateMouseAndMouseCursorPosition`, you can prevent mouse movement from changing the cursor position. Pass `true` to reverse the effect.

You can change the location of the mouse cursor on a specific display by calling the function `CGDisplayMoveCursorToPoint`. This function takes two parameters, a display ID and a point. The location of the point is relative to the origin or upper-left corner of the display. You can also change the location of the mouse cursor to any display by calling the function `CGWarpMouseCursorPosition`. This function takes a single parameter, a point in global display coordinates. Calling either of these functions does not generate a mouse event.

Listing 1 shows how you would hide the cursor, disassociate mouse movement from the cursor, move it to the origin of the main display, and then restore the cursor and mouse when you are done.

Listing 1 Controlling the mouse cursor

```
CGDisplayHideCursor (kCGNullDirectDisplay);
CGAssociateMouseAndMouseCursorPosition (false);
CGDisplayMoveCursorToPoint (kCGDirectMainDisplay, CGPointZero);
/* perform your application's main loop */
CGAssociateMouseAndMouseCursorPosition (true);
CGDisplayShowCursor (kCGNullDirectDisplay);
```

Document Revision History

This table describes the changes to *Quartz Display Services Programming Topics*.

Date	Notes
2006-06-28	New document that shows how to use Quartz Display Services to accomplish some basic tasks.