Technical Note TN2007
The CGDirectDisplay API

CONTENTS

This Technote provides a reference for the CGDirectDisplay API on Mac OS X.

The CGDirectDisplay API is designed to provide direct access to display modes, cursor position, gamma tables, and other low-level functionality. It is intended to replace and extend DrawSprocket, the Display Manager, and other similar APIs with one single, coherent, integrated API.

This Technote describes the important data types, two typical usage patterns, and all of the function calls in the API. A separate Technote will cover the API provided by CGDirectPalette.

Updated: [Nov 07 2000]

Definitions

Types used by the CGDirectDisplay API:

Data Type	Description
CGDirectDisplayID	An opaque reference to a display (kCGDirectMainDisplay refers to the main display as a convenience)
CGDirectPaletteRef	An opaque reference to a palette
CGDisplayCount	An unsigned 32 bit value
CGTableCount	An unsigned 32 bit value
CGDisplayCoord	A signed 32 bit value representing a coordinate on a display (The display origin is in the upper left corner)
CGByteValue	An unsigned 8 bit value used by the gamma/palette functions
CGOpenGLDisplayMask	An unsigned 32 bit value holding OpenGL display mask bits (Each bit represents a different display)
CGBeamPosition	An unsigned 32 bit value representing the position of the refresh beam on the display (0 is at the top)
CGMouseDelta	A signed 32 bit value representing the change in the mouse position
CGDisplayErr	A signed 32 bit error value

Back to top

Typical Usage

Simple:

In the simple case, you want to take over the main display and set it to the mode that most closely matches your required bit depth and resolution. You can then do any drawing you need to directly to the base address of the display. Once you are done, restore the display mode and release the display.

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

originalMode = CGDisplayCurrentMode( kCGDirectMainDisplay );

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

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

    CGDisplayCapture( kCGDirectMainDisplay );
    CGDisplaySwitchToMode( kCGDirectMainDisplay, mode );
    CGDisplayHideCursor( kCGDirectMainDisplay );

        /* Do your drawing/game loop here Use
        CGDisplayBaseAddress() to get the base
        address of the display */

    CGDisplayShowCursor( kCGDirectMainDisplay );
    CGDisplaySwitchToMode( kCGDirectMainDisplay, originalMode );
    CGDisplayRelease( kCGDirectMainDisplay );
}
```

Listing 1. Code listing for the simple case.

Complex:

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

```
CFDictionaryRef originalMode;
CGDirectDisplayID displays[kMaxDisplays];
CGDisplayCount numDisplays;
CGDisplayCount i;
CGDisplayErr err;
CFDictionaryRef bestMode = NULL;
CGDirectDisplayID bestDisplay = kCGDirectMainDisplay;

err = CGGetActiveDisplayList(
    kMaxDisplays,
    displays,
    &numDisplays);
if ( err != kCGDisplayNoErr )
{
    printf("Cannot get displays (%d)\n", err);
    exit( 1 );
}

for ( i = 0; i < numDisplays; i++ )
{
    CFDictionaryRef mode;
    CFIndex i, cnt;
    CGDirectDisplayID dspy;
    CFArrayRef modeList;
```

```
    dspy = displays[i];

    modeList = CGDisplayAvailableModes(dspy);
    if ( NULL != modeList )
    {
        //  Examine each mode
        cnt = CFArrayGetCount( modeList );

        for ( i = 0; i < cnt; i++ )
        {
            //  Pull the mode dictionary out of the CFArray
            mode = CFArrayGetValueAtIndex( modeList, i );

            /* Examine the mode here thisModeIsTheBest( mode )
            is a user supplied function that evaluates each
            mode and picks the best one (returning true) */
            if ( thisModeIsTheBest( mode ) )
            {
                bestMode = mode;
                bestDisplay = dspy;
            }
        }
    }
}

//  At this point, you have identified the best mode
//      and its corresponding display ID
if ( NULL != bestMode )
{
    originalMode = CGDisplayCurrentMode( bestDisplay );

    CGDisplayCapture( bestDisplay );
    CGDisplaySwitchToMode( bestDisplay, bestMode );
    CGDisplayHideCursor( bestDisplay );

    //  Do your drawing/game loop here
    //  Use CGDisplayBaseAddress() to get the base address of the display

    CGDisplayShowCursor( bestDisplay );
    CGDisplaySwitchToMode( bestDisplay, originalMode );
    CGDisplayRelease( bestDisplay );
    }
```

Listing 2. Code listing for the complex case.

Back to top

Finding Display IDs

```
CGDisplayErr CGGetActiveDisplayList(
    CGDisplayCount maxDisplays,
    CGDirectDisplayID * activeDspys,
    CGDisplayCount * dspyCnt);
```

- Lists all active displays.

```
CGDisplayErr CGGetDisplaysWithPoint(
    CGPoint point,
    CGDisplayCount maxDisplays,
    CGDirectDisplayID * dspys,
    CGDisplayCount * dspyCnt);
```

- Lists all displays containing `point`.
- Returns 0 in `dspyCnt` if no displays contain `point`.
- Note that displays can overlap (via mirroring, for example) so the `dspys` array may contain more than one display ID.

```
CGDisplayErr CGGetDisplaysWithRect(
    CGRect rect,
    CGDisplayCount maxDisplays,
    CGDirectDisplayID * dspys,
    CGDisplayCount * dspyCnt);
```

- Lists all displays containing at least part of `rect`.
- Returns 0 in `dspyCnt` if no displays contain `point`.
- Note that displays can overlap (via mirroring, for example) so the `dspys` array may contain more than one display ID.

Back to top

Selecting Display Modes

```
CFDictionaryRef CGDisplayCurrentMode(
    CGDirectDisplayID display);
```

- Returns a CFDictionaryRef describing the current display mode.
- Returns NULL for an invalid display ID.

```
CFArrayRef CGDisplayAvailableModes(
    CGDirectDisplayID display);
```

- Returns a CFArray of CFDictionaries describing all available modes.
- Returns NULL for an invalid display ID.

```
CFDictionaryRef CGDisplayBestModeForParameters(
    CGDirectDisplayID display,
    size_t bitsPerPixel,
    size_t width,
    size_t height,
    boolean_t *exactMatch);
```

- Finds a display mode of the specified depth with dimensions equal to or greater than those specified.
- If no exact depth match is found, checks for the next larger depth with dimensions equal to or greater than those specified.
- If no mode satisfies the criteria, returns the current mode.

- Sets `exactMatch` (if not `NULL`) to `true` if an exact match is found and `false` if no exact match is available.
- Returns NULL for an invalid display ID.

Back to top

Retrieving Display Mode Information

```
CGRect CGDisplayBounds(CGDirectDisplayID display);
```

- Returns the screen size and origin in global coordinates.
- Returns an empty rect for an invalid display ID.

```
size_t CGDisplayPixelsWide(
    CGDirectDisplayID display);
```

- Returns the display width in pixels.
- Returns 0 for an invalid display ID.

```
size_t CGDisplayPixelsHigh(
    CGDirectDisplayID display);
```

- Returns the display height in pixels.
- Returns 0 for an invalid display ID.

```
size_t CGDisplayBitsPerPixel(
    CGDirectDisplayID display);
```

- Returns the current bit depth of the specified display.
- Returns 0 for an invalid display ID.

```
size_t CGDisplayBitsPerSample(
    CGDirectDisplayID display);
```

- Returns the number of bits per color sample for the specified display.
- Returns 0 for an invalid display ID.

```
size_t CGDisplaySamplesPerPixel(
    CGDirectDisplayID display);
```

- Returns the number of color samples per pixel for the specified display.
- Returns 0 for an invalid display ID.

```
size_t CGDisplayBytesPerRow(
    CGDirectDisplayID display);
```

- Returns the number of bytes per row for the specified display.
- Returns 0 for an invalid display ID.

```
void * CGDisplayBaseAddress(
    CGDirectDisplayID display);
```

- Returns the base address of the specified display.
- Returns NULL for an invalid display ID.
- Note that if the display has not been captured, the returned address may refer to read-only memory.

```
void * CGDisplayAddressForPosition(
    CGDirectDisplayID display,
    CGDisplayCoord x,
    CGDisplayCoord y);
```

- Returns the address for location (X,Y) in screen coordinates.
- (0,0) represents the upper-left corner of the display.
- Returns NULL if the display ID is invalid or the X and Y coordinates are out of bounds.
- Note, if the display has not been captured, the returned address may refer to read-only memory.

Back to top

Using CGDirectDisplayIDs with OpenGL

```
CGDisplayErr CGGetDisplaysWithOpenGLDisplayMask(
    CGOpenGLDisplayMask mask,
    CGDisplayCount maxDisplays,
    CGDirectDisplayID * dspys,
    CGDisplayCount * dspyCnt);
```

- Finds a list of displays whose mask bits are specified in `mask`.
- Note that more than one display can be specified in the mask so the `dspys` array may contain more than one display ID.

```
CGOpenGLDisplayMask CGDisplayIDToOpenGLDisplayMask(
    CGDirectDisplayID display);
```

- Finds the OpenGL display mask corresponding to a given display ID.
- Returns 0 for an invalid display ID.

Back to top

Capturing and Releasing Displays

```
CGDisplayErr CGDisplayCapture(
    CGDirectDisplayID display);
```

- Captures the specified display to put up the shield window and allow for synchronous mode changes.

```
CGDisplayErr CGDisplayRelease(
    CGDirectDisplayID display);
```

- Releases the specified display and removes the shield window.

```
boolean_t CGDisplayIsCaptured(
    CGDirectDisplayID display);
```

- Returns `true` if the specified display is captured.

Back to top

Switching the Display Mode

```
CGDisplayErr CGDisplaySwitchToMode(
    CGDirectDisplayID display,
    CFDictionaryRef mode);
```

- Switches the specified display into the mode described by the `mode` dictionary.
- Note that switching modes may change display parameters and addresses.
- The selected display mode persists for the life of the program and, when the program terminates, the display mode automatically reverts to the mode set in the Monitors panel in the System Preferences.
- Note that display mode switching is only synchronous when applied to a captured display. If the display has not been captured, mode switching is asynchronous and may not be complete when `CGDisplaySwitchToMode` returns.

Back to top

Adjusting the Display Gamma and Palette

```
CGDisplayErr CGSetDisplayTransferByFormula(
    CGDirectDisplayID display,
    CGGammaValue redMin,
    CGGammaValue redMax,
    CGGammaValue redGamma,
    CGGammaValue greenMin,
    CGGammaValue greenMax,
    CGGammaValue greenGamma,
    CGGammaValue blueMin,
    CGGammaValue blueMax,
    CGGammaValue blueGamma);
```

- Sets the display gamma/transfer function from a formula specifying min and max values and a gamma for each channel.

- Gamma values must be greater than 0.0.
- Specify a value of (1.0 / 1.6) to get an antigamma of 1.6.
- Min values must be greater than or equal to 0.0 and less than 1.0.
- Max values must be greater than 0.0 and less than or equal to 1.0.
- Returns kCGSRangeCheck error if the values are out of range or max is greater than or equal to min.
- Values are computed by sampling a function for a range of indices from 0 through 1:
  value = Min + ((Max - Min) * pow(index, Gamma))
- The resulting values are converted to a machine specific format and loaded into the hardware.

```
CGDisplayErr CGGetDisplayTransferByFormula(
    CGDirectDisplayID display,
    CGGammaValue *redMin,
    CGGammaValue *redMax,
    CGGammaValue *redGamma,
    CGGammaValue *greenMin,
    CGGammaValue *greenMax,
    CGGammaValue *greenGamma,
    CGGammaValue *blueMin,
    CGGammaValue *blueMax,
    CGGammaValue *blueGamma);
```

- Gets the display gamma/transfer function from a formula specifying min and max values and a gamma for each channel.

```
CGDisplayErr CGSetDisplayTransferByTable(
    CGDirectDisplayID display,
    CGTableCount tableSize,
    const CGGammaValue  *redTable,
    const CGGammaValue  *greenTable,
    const CGGammaValue  *blueTable);
```

- Sets the display gamma/transfer function using tables of data for each channel.
- Values within each table should have values in the range of 0.0 through 1.0.
- The same table may be passed in for red, green, and blue channels.
- `tableSize` indicates the number of entries in each table.
- The tables are interpolated as needed to generate the number of samples needed by hardware.

```
CGDisplayErr CGGetDisplayTransferByTable(
    CGDirectDisplayID display,
    CGTableCount capacity,
    CGGammaValue  *redTable,
    CGGammaValue  *greenTable,
    CGGammaValue  *blueTable,
    CGTableCount *sampleCount);
```

- Gets the display transfer tables.
- `capacity` indicates the number of samples each array can hold.
- `sampleCount` is filled in with the number of samples actually copied in.

```
CGSetDisplayTransferByByteTable(
    CGDirectDisplayID display,
    CGTableCount tableSize,
    const CGByteValue  *redTable,
    const CGByteValue  *greenTable,
    const CGByteValue  *blueTable);
```

- Sets the display gamma/transfer function using tables of byte values for each channel.
- Values within each table should have values in the range of 0 through 255.
- The same table may be passed in for red, green, and blue channels.
- `tableSize` indicates the number of entries in each table.
- The tables are interpolated as needed to generate the number of samples needed by hardware.

```
boolean_t CGDisplayCanSetPalette(
    CGDirectDisplayID display);
```

- Returns `true` if the specified display supports palettes.
- 8-bit pseudo-color only.

```
CGDisplayErr CGDisplaySetPalette(
    CGDirectDisplayID display,
    const CGDirectPaletteRef palette);
```

- Sets the palette for the specified display ID.
- Note that the current gamma function is applied to the palette elements before they are loaded into the hardware.

Back to top

Accessing the CoreGraphics Shielding Window

```
void * CGShieldingWindowID(
    CGDirectDisplayID display);
```

- Returns the CoreGraphics raw shield window ID for use with the drawing surface APIs.
- Returns NULL if the display ID is invalid or the display is not shielded.

```
int32_t CGShieldingWindowLevel(void);
```

- Returns the window level used for the shielding window.
- This value may be used to position Cocoa windows at the same window level as the shielding window.

Back to top

Controlling the Mouse Cursor

```
CGDisplayErr CGDisplayHideCursor(
    CGDirectDisplayID display);
```

- Increments the hide cursor count, hiding the mouse cursor if necessary.

```
CGDisplayErr CGDisplayShowCursor(
    CGDirectDisplayID display);
```

- Decrements the hide cursor count, showing the mouse cursor if necessary.

```
CGDisplayErr CGDisplayMoveCursorToPoint(
    CGDirectDisplayID display, CGPoint point);
```

- Moves the mouse cursor to the specified point relative to the display origin (the upper left corner of the display).
- Returns `kCGDisplayNoErr` if successful.
- Note that no events are generated as a result of this move.
- Points that would lie outside the desktop are clipped to the desktop.

```
void CGGetLastMouseDelta(
    CGMouseDelta * deltaX,
    CGMouseDelta * deltaY );
```

- Returns the mouse position change associated with the last mouse move event received by this application.

Back to top

Waiting for the Vertical Blank

```
CGDisplayErr CGDisplayWaitForBeamPositionOutsideLines(
    CGDirectDisplayID display,
    CGBeamPosition upperScanLine,
    CGBeamPosition lowerScanLine );
```

- Waits until the beam position is outside the range specified by `upperScanLine` and `lowerScanLine`.
- Returns `kCGDisplayNoErr` on success and an error if `display`, `upperScanLine`, or `lowerScanLine` are invalid.
- Note that if `upperScanLine` and `lowerScanLine` encompass the entire display height, the function returns an error.
- `lowerScanLine` must be greater than or equal to `upperScanLine`.
- Some display systems may not use conventional video vertical and horizontal sweeps in refreshing the display. These displays report a `kCGDisplayRefreshRate` of 0 in the CFDictionaryRef returned by `CGDisplayCurrentMode` and on such displays `CGDisplayWaitForBeamPositionOutsideLines` returns at once.
- `upperScanLine` and `lowerScanLine` should be set to allow enough lead time for the drawing operation to complete. A common strategy is to wait for the beam to pass the bottom of the drawing area, allowing almost a full vertical sweep period to perform drawing. To do this, set `upperScanLine` to 0, and set `lowerScanLine` to the bottom of the bounding box:
  `lowerScanLine = (CGBeamPosition)(cgrect.origin.y + cgrect.size.height);`

```
CGBeamPosition CGDisplayBeamPosition(
    CGDirectDisplayID display );
```

- Returns the current beam position on the display.
- Returns 0 if `display` is invalid or the display doesn't implement a conventional vertical sweep for painting.

Back to top

Summary

This Technote covers the basic information you need to start using the CGDirectDisplay API, including the important type definitions, function calls, and typical usage patterns. Information on using the CGDirectPalette API will follow in an upcoming Technote.

Back to top

Downloadables

Acrobat version of this Note (124K). Download

Back to top
