Seed Draft

Manipulating Displays Using DrawSprocket

For DrawSprocket 1.7

Preliminary

October 20, 1999
Technical Publications

Introduction

IMPORTANT

This is a preliminary document. Although it has been reviewed for technical accuracy, it is not final. Apple Computer, Inc. is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. You can check <http://developer.apple.com/techpubs/macos8/SiteInfo/whatsnew.html> for information about updates to this and other developer documents. To receive notification of documentation updates, you can sign up for ADC's free Online Program and receive their weekly Apple Developer Connection News e-mail newsletter. (See <http://developer.apple.com/membership/index.html> for more details about the Online Program.) ▲

DrawSprocket is a subset of Apple Game Sprockets that gives your application control over special display features. It can interact with Mac OS system software as well as with specialized video subsystems and third-party video cards. For example, you can use DrawSprocket to choose a display resolution and pixel depth, perform gamma fading, and handle display buffering.

This document assumes you are familiar with programming Macintosh computers. It does not discuss Macintosh graphics systems or drawing functions, nor does it discuss video hardware. For more information on these topics you can consult *Inside Macintosh: Imaging with QuickDraw* and *Designing PCI Cards and Drivers for Power Macintosh Computers* respectively.

If you are building a game, you may also want to consult other Game Sprocket documentation:

- *Configuring Game Input Devices with InputSprocket*
- *Simplifying Networked Gaming Using NetSprocket*
- SoundSprocket documentation (forthcoming)

This document currently covers InputSprocket in the following chapters:

- Chapter 2, “DrawSprocket Reference,” contains a complete programming reference, documenting the functions, data types, and constants available with DrawSprocket.
- Appendix A, “Document Version History,” describes changes made from previous versions of DrawSprocket documentation.

For additional information about creating games for the Macintosh, you should check the Apple Developer games Web site:

<http://developer.apple.com/games/>

C H A P T E R

DrawSprocket Reference

Contents

DrawSprocket Reference

This chapter describes the DrawSprocket application programming interface (API) introduced with InputSprocket 1.7. This chapter contains the following sections:

Note
This document describes version 1.7 of DrawSprocket. For a list of functions changed or added between versions 1.0 and 1.7, see Appendix A. ◆

DrawSprocket Functions

This section describes DrawSprocket functions in the following categories:

Testing for the Availability of DrawSprocket

To determine whether the DrawSprocket library is available, you should check for resolved symbol addresses before calling any DrawSprocket functions. For example, you could use code similar to the following:

```
// Check to see if the Code Fragment Manager has resolved
// DrawSprocket symbols
if ((Ptr) DSpStartup == (Ptr) kUnresolvedCFragSymbolAddress)
    {
        // Post error message here
        ExitToShell();
    }
```

Activating and Deactivating DrawSprocket

You use the functions in this section before using DrawSprocket and when you are finished.

- `DSpGetVersion` (page 10) determines the version of DrawSprocket installed on the host computer.
- `DSpStartup` (page 11) initializes DrawSprocket.
- `DSpShutdown` (page 11) shuts down DrawSprocket.

DSpGetVersion

Determines the version of DrawSprocket installed on the host computer.

```
NumVersion DSpGetVersion (void);
```

function result The version number of DrawSprocket installed.

VERSION NOTES

Introduced with DrawSprocket 1.7.

DSpStartup

Initializes DrawSprocket

```
OSStatus DSpStartup (void);
```

function result A result code. See “Result Codes” (page 87).

DISCUSSION

You must call this function before attempting to call any DrawSprocket functions (except for `DSpGetVersion` (page 10)).

Note that the debug version of DrawSprocket will notify you if you did not call `DSpStartup`.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpShutdown

Shuts down DrawSprocket

```
OSStatus DSpShutdown  (void);
```

function result A result code. See “Result Codes” (page 87).

DISCUSSION

You must call this function before quitting the application.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Choosing a Context

The functions in this section allow you to determine the display characteristics and features of a given system and help you choose the configuration that best fits your game's needs.

- `DSpFindBestContext` (page 13) finds the context that best matches the requirements you specify.
- `DSpFindBestContextOnDisplayID` (page 14) determines the best context to use for a given display.
- `DSpContext_GetDisplayID` (page 14) obtains the ID of the display a context is associated with.
- `DSpGetCurrentContext` (page 15) obtains a reference to the current display context for a given display.
- `DSpGetFirstContext` (page 16) obtains the first context in the list of contexts available for a specified display.
- `DSpGetNextContext` (page 17) obtains the next context in a list of available contexts for a display.
- `DSpContext_GetAttributes` (page 18) obtains the attributes of a context as if it were in the active state.
- `DSpCanUserSelectContext` (page 19) determines whether there is a meaningful choice of contexts to present to the user with the `DSpUserSelectContext` function.
- `DSpUserSelectContext` (page 20) presents a dialog box that allows the user to select a display.

DSpFindBestContext

Finds the context that best matches the requirements you specify.

```
OSStatus DSpFindBestContext (
                    const DSpContextAttributesPtr inDesiredAttributes,
                    DSpContextReference *outContext);
```

`inDesiredAttributes`
A pointer to a context attributes structure describing the desired display characteristics of the context, such as display height and width, preferred pixel depth, and color capability. See `DSpContextAttributes` (page 65) for more information about this structure.

`outContext` On return, a reference to the context that best meets or exceeds the specified attribute requirements, or `NULL` if no such context exists.

function result A result code. If no context meets the requirements you specified, the function returns `kDSpContextNotFoundErr`. See “Result Codes” (page 87) for additional return values.

DISCUSSION

Even if the call to `DSpFindBestContext` returns successtully, the game should check the attributes of the chosen context by calling the function `DSpContext_GetAttributes` (page 18). It is possible that the game may want to use attributes of the context that exceed those asked for. For example, the game may request a mode such as 320x200x8 but the best match is a 640x480x8 display; the game can adapt to a full screen mode once it is aware of the situation.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpFindBestContextOnDisplayID

Determines the best context to use for a given display.

```
OSStatus DSpFindBestContextOnDisplayID (
                     DSpContextAttributesPtr  inDesiredAttributes,
                     DSpContextReference *outContext,
                     DisplayIDType inDisplayID);
```

`inDesiredAttributes`	A pointer to a structure describing the desired attributes for the context. See `DSpContextAttributes` (page 65) for more information.
`outContext`	On return, `outContext` points to the context that best matches the desired attributes, or `NULL` if no such context exists.
`inDisplayID`	The ID of the display to check for contexts.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

You can obtain the display ID of a monitor by calling the Display Manager.

VERSION NOTES

Introduced with DrawSprocket 1.7.

DSpContext_GetDisplayID

Obtains the ID of the display a context is associated with.

```
OSStatus DSpContext_GetDisplayID (
                     DSpContextReference inContext,
                     DisplayIDType *outDisplayID);
```

`inContext`	A reference to the context whose monitor display ID you want to determine.

`outDisplayID`	On return, the display ID for the monitor associated with the context.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

Note that 3D hardware accelerators (such as RAVE) typically must draw using a graphics device (`GDevice`) rather than a graphics port. To do so, you can call `DSpContext_GetDisplayID` to get the display ID of the device associated with the context and then call the Display Manager function `DMGetDeviceByDisplayID` to obtain the `GDevice`.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpGetCurrentContext

Obtains a reference to the current display context for a given display.

```
OSStatus DSpGetCurrentContext (
                    DisplayIDType inDisplayID,
                    DSpContextReference *outContext);
```

`inDisplayID`	The ID of the display whose context you want to obtain.
`outContext`	On return, `outContext` points to the current context in the given display.
function result	A result code. See “Result Codes” (page 87).

VERSION NOTES

Introduced with DrawSprocket 1.7.

DSpGetFirstContext

Obtains the first context in the list of contexts available for a specified display.

```
OSStatus DSpGetFirstContext (
                    DisplayIDType displayID,
                    DSpContextReference *outContext);
```

`displayID`	The ID of the display whose context you desire. You can obtain the display ID by calling the Display Manager.
`outContext`	On return, a reference to the first context in the list of available contexts for the specified display. You cannot use this context with any function other than `DSpContext_GetAttributes`, `DSpContext_GetFlattendSize`, `DSpContext_Flatten`, and `DSpContext_GetDisplayID` unless you reserve it with `DSpContext_Reserve`.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

Using the function `DSpGetFirstContext` in combination with `DSpGetNextContext` (page 17) allows you to iterate over the list of contexts and choose one that best suits your needs. You may also have DrawSprocket find one for you with `DSpFindBestContext` or let the user select one by calling `DSpUserSelectContext`.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpGetNextContext

Obtains the next context in a list of available contexts for a display.

```
OSStatus DSpGetNextContext (
                     DSpContextReference inCurrentContext,
                     DSpContextReference *outContext);
```

`inCurrentContext`
: A reference to a context in the list of contexts available for a display. This should be a reference that was just returned by `DSpGetFirstContext` or `DSpGetNextContext`. If this parameter contains the last context in the list, `DSpGetNextContext` returns an error.

`outContext`
: On return, a reference to the next context in the list of available contexts.

function result
: A result code. See “Result Codes” (page 87).

DISCUSSION

Using the function `DSpGetNextContext` in combination with `DSpGetFirstContext` (page 16) allows you to iterate over the list of contexts and choose one that best suits your needs. For example, you could have code such as the following:

```
DSpContextReference theContext;

theError = DSpGetFirstContext(theDisplayID, &theContext);
/* process the error */
while (theContext)
{
    /* process the context */

    /* get the next context */
    theError = DSpGetNextContext(theContext, &theContext);
    /* process the error */
}
```

You may also have DrawSprocket find a display context for you by calling `DSpFindBestContext` (page 13) or `DSpFindBestContextOnDisplayID` (page 14), or let the user select one by calling `DSpUserSelectContext` (page 20).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetAttributes

Obtains the attributes of a context as if it were in the active state.

```
OSStatus DSpContext_GetAttributes (
                    DSpContextReference inContext,
                    DSpContextAttributesPtr outAttributes);
```

`inContext`	The context whose attributes you want to get.
`outAttributes`	On return, a pointer to an attributes structure describing the context. See `DSpContextAttributes` (page 65) for more information about this structure.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

You can use this function to confirm that the context returned from `DSpFindBestContext` (page 13) has the characteristics you need. You may even adjust your drawing plans based on the results. For example, you might have requested a resolution mode such as 320x200x8 when calling `DSpFindBestContext`, but then learned from calling `DSpContext_GetAttributes` that the context is a 640x480x8 display. In such a case, you might still use the 640x480x8 display, but display a larger game image.

Note that the monitor frequency may not be known until a context is actually in the active play state, so it may return as zero.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpCanUserSelectContext

Determines whether there is a meaningful choice of contexts to present to the user with the `DSpUserSelectContext` function.

```
OSStatus DSpCanUserSelectContext  (
                    DSpContextAttributesPtr inDesiredAttributes
                    Boolean *outUserCanSelectContext );
```

`DSpContextAttributesPtr`
: A pointer to a context attributes structure that specifies the required attributes. See `DSpContextAttributes` (page 65) for more information.

`outUserCanSelectContext`
: On return, the value is `true` if there are multiple contexts that meet the specified attribute requirements; `false` if there are not.

function result A result code. See “Result Codes” (page 87).

DISCUSSION

This function `DSpCanUserSelectContext` allows you to check whether calling `DSpUserSelectContext` is useful so as to avoid presenting the user with a selection dialog box when there is no choice of displays.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpUserSelectContext

Presents a dialog box that allows the user to select a display.

```
OSStatus DSpUserSelectContext (
                    DSpContextAttributesPtr inDesiredAttributes,
                    DisplayIDType inDialogDisplayLocation,
                    DSpEventProcPtr inEventProc,
                    DSpContextReference *outContext );
```

`inDesiredAttributes`
: A pointer to an attributes structure that specifies a minimum set of required display characteristics. See `DSpContextAttributes` (page 65) for more information.

`inDialogDisplayLocation`
: The ID of the display on which to present the selection dialog box. If this parameter is 0, DrawSprocket positions the dialog box on the main screen.

`inEventProc`
: A pointer to an application-defined event-processing function that allows you to handle events received by the dialog box that DrawSprocket cannot process, such as update events, in your game context area. See the function `MyEventHandler` (page 63) for more information about implementing this function.

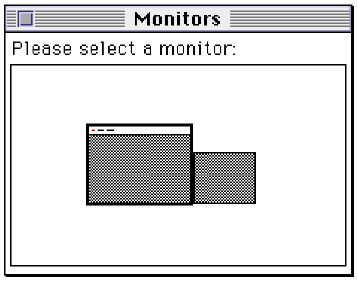
`outContext`
: On return, a reference to a context.

function result
: A result code. See “Result Codes” (page 87).

DISCUSSION

In the selection dialog box (Figure 2-1), all graphics devices appear, although the user can select only those contexts that meet or exceed the minimum characteristics given in the `inDesiredAttributes` parameter.

Figure 2-1 A context-selection dialog box



SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Saving and Restoring a Context

The functions in this section allow you to flatten drawing contexts to be saved (for example, to the game's preferences file) and restore them at a later time.

- `DSpContext_Restore` (page 22) restores a context that was saved previously, most likely to preserve a user's preferences.
- `DSpContext_GetFlattenedSize` (page 23) determines how much memory is required to store a flattened version of a context.
- `DSpContext_Flatten` (page 23) converts a context into a format suitable for saving to disk—for example, to save user preferences.

DSpContext_Restore

Restores a context that was saved previously, most likely to preserve a user's preferences.

```
OSStatus DSpContext_Restore (
                    void *inFlatContext,
                    DSpContextReference *outRestoredContext);
```

`inFlatContext`
A pointer to the flattened context. Typically, the context would have been saved out to disk and reloaded on a later execution of the game before calling this function.

`outRestoredContext`
On return, a reference to the restored context, if it exists.

function result A result code. See "Result Codes" (page 87).

DISCUSSION

If `DSpContext_Restore` can't find a match, the user probably has reconfigured the displays since the last time your game was run, and the call returns an error. This function has a high probability of failure, so your game should not rely on being able to restore the context. However, the game should attempt to do so as part of the normal saving of the user preferences.

If you save a context, flatten it by calling `DSpContext_Flatten` (page 23) before you first make the context's play state active; otherwise, the saved data will not contain the proper information with which to locate the display.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetFlattenedSize

Determines how much memory is required to store a flattened version of a context.

```
OSStatus DSpContext_GetFlattenedSize (
                     DSpContextReference inContext,
                     UInt32 *outFlatContextSize);
```

`inContext` A reference to the context you intend to flatten.

`outFlatContextSize`
On return, the number of bytes required to store a flattened version of the context.

function result A result code. See “Result Codes” (page 87).

DISCUSSION

After calling the `DSpContext_GetFlattenedSize` function, you can then allocate a buffer of `outFlatContextSize` size and pass it to `DSpContext_Flatten` (page 23).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_Flatten

Converts a context into a format suitable for saving to disk—for example, to save user preferences.

```
OSStatus DSpContext_Flatten (
                     DSpContextReference inContext,
                     void *outFlatContext);
```

`inContext`	A reference to the context to be flattened.
`outFlatContext`	A pointer to the buffer to hold the flattened context. The buffer must be large enough to hold the flattened context. You can find out the correct size by calling `DSpContext_GetFlattenedSize` (page 23). On return, the buffer holds the flattened context.
function result	A result code. See "Result Codes" (page 87).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Manipulating a Context

The functions in this section allow you to reserve a context, set the play state for a context, and set the color of the blanking window.

- `DSpContext_Reserve` (page 25) reserves a context so that you can begin using it in your game.
- `DSpContextQueue` (page 26) queues a context you want to switch to.
- `DSpContextSwitch` (page 27) switches display contexts.
- `DSpContext_Release` (page 28) releases a context you are finished using.
- `DSpContext_SetState` (page 28) sets the play state of a context.
- `DSpContext_GetState` (page 30) finds out the current play state of a context.
- `DSpSetBlankingColor` (page 30) assigns a background color to the blanking window for all displays.

DSpContext_Reserve

Reserves a context so that you can begin using it in your game.

```
OSStatus DSpContext_Reserve (
                    DSpContextReference inContext,
                    const DSpContextAttributesPtr inDesiredAttributes);
```

`inContext`	A reference to the context to reserve. When the context is reserved, it is in the inactive state. There will be no visible indication that the context has been reserved at this point. To enable your context, call `DSpContext_SetState` (page 28). The context will show up on the display once the context has been placed in the active state.
`inDesiredAttributes`	A pointer to an attributes structure that specifies the configuration you would like for the display when it is in the active or paused state. If you would like to override the attributes of the context, you may do so in the attributes structure. For example, if you ask for a 320x240x16 display but the closest match is a context that is 640x480x32, passing in your requested attributes when you reserve the context will cause the `DSpContext_GetBackBuffer` function to return a graphics pointer that refers to a 320x240x16 drawing environment.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

You should turn off features that you are not interested in when you reserve the context. For example, if the context supports page flipping (and you know this because you requested the actual capabilities of the context using `DSpContext_GetAttributes`), you can turn off the page-flipping bit in your desired attributes so that you will be assured of using software buffering.

You should only specify a back buffer bit depth different from the display bit depth when you absolutely must, as it is the worst case scenario for DrawSprocket and will result in a synchronous call to `CopyBits` to bring your back buffer to the display.

To release a reserved context, you must call the function `DSpContext_Release` (page 28).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContextQueue

Queues a context you want to switch to.

```
OSStatus DSpContext_Queue (
                      DSpContextReference inParentContext,
                      DSpContextReference inChildContext,
                      DSpContextAttributesPtr inDesiredAttributes);
```

`inParentContext`
The current active context.

`inChildContext`
The context you want to switch to.

`inDesiredAttributes`
A pointer to a context attributes structure that describes the context you want to switch to.

function result A result code. See “Result Codes” (page 87).

DISCUSSION

Typically, you use this function to queue up contexts in addition to the one specified by the function `DSpContext_Reserve` (page 25). After you queue a context, you make it active by calling the function `DSpContextSwitch` (page 27). To release a queued context, you must call the function `DSpContext_Release` (page 28).

Calling `DSpContext_Queue` also determines whether the desired context switch is actually possible. For example, among other things, DrawSprocket will check to see that both contexts are on the same display. If the contexts are incompatible, this call returns an error.

Note that you can also use this function to modify attributes of the context to be switched to.

VERSION NOTES

Introduced with DrawSprocket 1.7.

DSpContextSwitch

Switches display contexts.

```
OSStatus DSpContext_Switch (
                    DSpContextReference inOldContext,
                    DSpContextReference inNewContext);
```

`inOldContext`	The current display context.
`inNewContext`	The display context to switch to.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

Calling this function switches the display context immediately without any intermediate switch to the default display mode. Note that switching contexts will kill any piggyback VBL routines attached to the context you are switching out.

If you did not queue the contexts you want to switch (by calling the function `DSpContextQueue` (page 26)), `DSpContextSwitch` returns an error.

VERSION NOTES

Introduced with DrawSprocket 1.7.

DSpContext_Release

Releases a context you are finished using.

```
OSStatus DSpContext_Release  (DSpContextReference inContext);
```

`inContext`	A reference to the context to be released. Releasing the context does not necessarily remove the blanking window from the corresponding display. All displays remain covered by the blanking window until all contexts have been released or put in an inactive play state.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

You must release the context whether it was reserved or queued.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_SetState

Sets the play state of a context.

```
OSStatus DSpContext_SetState (
                    DSpContextReference inContext,
                    DSpContextState inState);
```

`inContext`	A reference to the context whose play state you want to set.

`inState`	A constant specifying the desired play state. Valid input values for this parameter are `kDSpContextState_Active`, `kDSpContextState_Paused`, and `kDSpContextState_Inactive`. See "Play State Constants" (page 75) for more information.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

In summary, you can make these choices:

- A context's initial play state is inactive. When all contexts for a display are set to `kDSpContextState_Inactive`, the display looks exactly as it does when the user is using their Macintosh normally: the monitor resolutions are set to the default, the menu bar is available, and so on.
- Set the play state to `kDSpContextState_Active` to use the display. In this state, the attributes of the context are used to change the display resolution, remove the menu bar, and so on. When at least one context is active, all the display devices in the system are covered by a blanking window. When a context is in the active state, the display is completely owned by the game.
- Set the play state to `kDSpContextState_Paused` to temporarily restore system adornments, while maintaining the attributes used by the context. This gives the user the opportunity to use the menus and switch to other applications. While the context is in the paused state, it is very important to call `DSpProcessEvent` to allow DrawSprocket to correctly handle events such as suspend or resume (see `DSpProcessEvent` (page 58). Page flipping and double buffering are inactive in this state, and the context will be placed back at page 0 if page flipping was being used.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetState

Determines the current play state of a context.

```
OSStatus DSpContext_GetState (
                        DSpContextReference inContext,
                        DSpContextState *outState);
```

`inContext`	A reference to the context whose play state you want to get.
`outState`	On return, the play state of the context. Valid return values are `kDSpContextState_Active`, `kDSpContextState_Paused`, and `kDSpContextState_Inactive`. See “Play State Constants” (page 75) for more information.
function result	A result code. See “Result Codes” (page 87).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpSetBlankingColor

Assigns a background color to the blanking window for all displays.

```
OSStatus DSpSetBlankingColor  (const RGBColor *inRGBColor);
```

`inRGBColor`	A pointer to the background color to use for the blanking window.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

The blanking color replaces the desktop and system adornments, such as the menu bar, for all display devices as long as any context is active.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Drawing and Double Buffering

The functions in this section allow you to draw to the display and control various aspects of display visibility and frame speed.

- `DSpContext_FadeGamma` (page 32) sets the brightness of the display to the specified intensity.
- `DSpContext_FadeGammaOut` (page 34) completely fades out a display to a color of your choice.
- `DSpContext_FadeGammaIn` (page 35) completely fades in a display to a color of your choice.
- `DSpContext_GetFrontBuffer` (page 36) obtains the front buffer for the context.
- `DSpContext_GetBackBuffer` (page 37) obtains the back buffer for the context.
- `DSpContext_InvalBackBufferRect` (page 38) invalidates a specific area of a context's back buffer, so that only a portion of the screen needs to be redrawn when the buffers are next swapped.
- `DSpContext_SwapBuffers` (page 39) draws a context's back buffer to the screen.
- `DSpContext_IsBusy` (page 40) finds out whether a back buffer is available.
- `DSpContext_SetDirtyRectGridSize` (page 41) suggests a grid size for the context's dirty rectangles.
- `DSpContext_GetDirtyRectGridSize` (page 42) finds out the current grid size for a context's dirty rectangles.

- `DSpContext_GetDirtyRectGridUnits` (page 43) finds out the size of the base dirty rectangle grid for a context.
- `DSpContext_SetMaxFrameRate` (page 44) sets a maximum frame rate for a specified context.
- `DSpContext_GetMaxFrameRate` (page 45) obtains the maximum frame rate for a specified context.
- `DSpContext_GetMonitorFrequency` (page 45) obtains the frequency for the display associated with a context.

DSpContext_FadeGamma

Sets brightness of the display to the specified intensity.

```
OSStatus DSpContext_FadeGamma (
                    DSpContextReference inContext,
                    SInt32 inPercentOfOriginalIntensity,
                    RGBColor *inZeroIntensityColor);
```

`inContext`	A reference to the context whose display is to be faded. If you pass `NULL` for this parameter, the fade operation applies simultaneously to all displays.
`inPercentOfOriginalIntensity`	The percentage (0–100) of the display’s full intensity that you want to achieve with this call. Values above 100 percent begin to converge on white. If you have specified an intensity color, values less than zero begin to converge on black.
`inZeroIntensityColor`	A pointer to the color that is to correspond to zero intensity (represented by a value of 0 in the `inPercentOfOriginalIntensity` parameter). If you pass `NULL` for this parameter, the zero-intensity color is black.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

Fading the display is an aesthetically pleasing way to transition into and out of your game and between different sections of it. When performing a resolution-mode switch (as when activating and deactivating your context's play state), it is important to fade the display to hide the flash that occurs.

`DSpContext_FadeGamma` performs a gamma fade, which gives better results than a simple indexed fade.

Fading using `DSpContext_FadeGamma` is an incremental process. That is, over a period of time, you make repeated, timed calls to `DSpContext_FadeGamma`, each time passing it an incrementally different value for the `inPercentOfOriginalIntensity` parameter, until the final desired intensity is achieved. The intensity value you pass is usually an integer between 0 and 100. It can be greater than 100, if you want to use fading to create a high-intensity burst of light, or less than 100 if you have specified a zero-intensity color and want to fade the color toward black.

The zero-intensity value that you fade out to is by default black, but it can be any color that you specify in the `inZeroIntensityColor` parameter. You can achieve special effects by fading partially toward one zero-intensity color and then completing the fade to a different one. At the point when you actually switch resolution modes, the zero-intensity color must be black and your display must be completely faded if there is to be no visible flash.

To automatically accomplish a smooth fade all the way from full intensity to zero intensity, or vice versa, in a single operation, use the `DSpContext_FadeGammaIn` function (page 35) and the `DSpContext_FadeGammaOut` (page 34) function.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_FadeGammaOut

Completely fades out a display to a color of your choice.

```
OSStatus DSpContext_FadeGammaOut (
                        DSpContextReference inContext,
                        RGBColor *inZeroIntensityColor);
```

`inContext`	A reference to the context whose display is to be faded. The function fades the display from 100 percent to 0 percent intensity over a period of one second. If you pass `NULL` for this parameter, the fade operation applies simultaneously to all displays.
`inZeroIntensityColor`	A pointer to the color that is to correspond to zero intensity. If you pass `NULL` for this parameter, the zero-intensity color is black.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

The initial gamma is that set by DrawSprocket when `DSpStartup` (page 11) was called, or the last gamma value set by calling the `DSpContext_FadeGamma` (page 32) function. If you had changed the system gamma to a different value, you may see a flash at the beginning of the fade due to the change in the initial gamma.

A key press or a mouse button click will jump the fade to its end point immediately.

You can perform a manual fade with the `DSpContext_FadeGamma` function (page 32).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_FadeGammaIn

Completely fades in a display to a color of your choice.

```
OSStatus DSpContext_FadeGammaIn (
                        DSpContextReference inContext,
                        RGBColor *inZeroIntensityColor);
```

`inContext`	A reference to the context whose display is to be faded. The function fades the display from 0 percent to 100 percent intensity over a period of one second. If you pass `NULL` for this parameter, the fade operation applies simultaneously to all displays.
`inZeroIntensityColor`	The color that is to correspond to zero intensity. If you pass `NULL` for this parameter, the zero-intensity color is black.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

A key press or a mouse-button click will jump the fade to its end point immediately.

You can perform a manual fade with the `DSpContext_FadeGamma` function (page 32).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetFrontBuffer

Obtains the front buffer for the context.

```
OSStatus DSpContext_GetFrontBuffer (
                        DSpContextReference inContext,
                        CGrafPtr *outBackBuffer);
```

`inContext`	A reference to the context whose front buffer is to be returned.
`outFrontBuffer`	On return, a pointer to the front buffer (that is, to a `CGrafPort`).
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

The front buffer is the screen display. Typically you use this function when you are not using backbuffers and you want to pass a `CGrafPtr` so another interface can draw to the screen (for example, by using OpenGL or QuickTime, or you simply want to change resolutions). However, if you are drawing to the screen yourself, you must call `DSpContext_GetFrontBuffer` each time through your game’s drawing cycle to compensate for possible page flipping.

Note
Note that 3D hardware accelerators (such as RAVE) typically must draw using a graphics device (`GDevice`) rather than a graphics port. To do so, you should call `DSpContext_GetDisplayID` (page 14) to get the display ID of the device the context is on and then call the Display Manager function `DMGetDeviceByDisplayID` to obtain the `GDevice`. ◆

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.1.2.

DSpContext_GetBackBuffer

Obtains the back buffer for the context.

```
OSStatus DSpContext_GetBackBuffer (
                        DSpContextReference inContext,
                        DSpBufferKind inBufferKind,
                        CGrafPtr *outBackBuffer);
```

`inContext`	A reference to the context whose back buffer is to be returned.
`inBufferKind`	The kind of buffer. Currently the only supported buffer kind is `kDSpBufferKind_Normal`.
`outBackBuffer`	On return, a pointer to the back buffer (that is, to a `CGrafPort`).
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

The back buffer, which is where the game should draw to, is the next buffer that will be displayed on a call to `DSpContext_SwapBuffers` (page 39).

The pointer to the back buffer may change after a call to `DSpContext_SwapBuffers`, so you must call this function before rendering every frame.

If you have specified an underlay for the context, the back buffer will have the underlay image restored before this call returns.

If there are no available back buffers (they are all queued up for display), this function will block until one is available. To avoid blocking, call `DSpContext_IsBusy` (page 40) until it returns `false`.

Note that 3D hardware accelerators typically must draw using a graphics device (`GDevice`) rather than a graphics port.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_InvalBackBufferRect

Invalidates a specific area of a context's back buffer, so that only a portion of the screen needs to be redrawn when the buffers are next swapped.

```
OSStatus DSpContext_InvalBackBufferRect (
                     DSpContextReference inContext,
                     const Rect *inRect);
```

`inContext`	A reference to the context whose back buffer is to be invalidated.
`inRect`	A pointer to a rectangle specifying the area (in back-buffer coordinates) to invalidate.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

If you do not call this function between buffer swaps, the entire back buffer is considered invalid when a swap occurs. The invalid rectangles must be set prior to each call to `DSpContext_SwapBuffers`; the dirty rectangle list is emptied before `DSpContext_GetBackBuffer` returns the back buffer for re-use.

You can make multiple calls to this function between swaps to accumulate invalid rectangular areas.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_SwapBuffers

Draws a context's back buffer to the screen.

```
OSStatus DSpContext_SwapBuffers(DSpContextReference inContext,
                      DSpCallbackProcPtr inBusyProc,
                      void *inUserRefCon);
```

`inContext`	A reference to the context whose buffers are to be swapped. The function causes the invalid parts of the back buffer of the context specified in this parameter (or the entire back buffer, if its invalid-rectangle list is empty) to be drawn to the screen.
`inBusyProc`	A pointer to an application-defined function that performs any required pre-swap tasks.
`inUserRefCon`	A reference constant to be handed back by DrawSprocket when it calls the callback specified by the `inBusyProc` parameter.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

This function returns immediately, even if the buffer swap has not yet occurred. To determine when the next call to `DSpContext_GetBackBuffer` will not block, you can repeatedly call the `DSpContext_IsBusy` function (page 40) until it returns a value of `false`.

Before performing the buffer swap, DrawSprocket repeatedly calls an application-supplied callback function, pointed to by the `inBusyProc` parameter, to make sure that any constraints you impose are satisfied before the swap occurs. When DrawSprocket calls the callback routine, it passes the reference constant you passed to `DspContext_SwapBuffers` in the `refCon` parameter.

See the function `MyCallbackFunction` (page 62) and the data type `DSpCallbackProcPtr` (page 70) for more information.

In a worst case scenario where the back buffer and the display have different bit depths, `DSpContext_SwapBuffers` immediately calls `CopyBits` to transfer the data. To avoid this, and to use the optimized DrawSprocket blitters, always insure that your back buffer and display bit depths are identical.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_IsBusy

Finds out whether a back buffer is available.

```
OSStatus DSpContext_IsBusy (
                    DSpContextReference inContext,
                    Boolean *outBusyFlag);
```

`inContext`	A reference to the context associated with the desired back buffer.
`outBusyFlag`	On return, contains `true` if no back buffer is available, `false` if a back buffer is available.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

You can use this function to determine whether a call to `DSpContext_GetBackBuffer` (page 37) will block.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_SetDirtyRectGridSize

Suggests a grid size for the context's dirty rectangles.

```
OSStatus DSpContext_SetDirtyRectGridSize (
                        DSpContextReference inContext,
                        UInt32 inCellPixelWidth,
                        UInt32 inCellPixelHeight);
```

`inContext`	A reference to a context whose dirty rectangle grid size you want to set.
`inCellPixelWidth`	The width of the grid in pixels.
`inCellPixelHeight`	The height of the grid in pixels.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

The `DSpContext_SetDirtyRectGridSize` function takes a reference to a context in the `inContext` parameter and sets the dirty rectangle grid size for that context as closely as possible to the dimensions passed in the `inCellPixelWidth` and `inCellPixelHeight` parameters. The size used depends on factors such as the L1 cache size and the CPU bus width, so your suggested values may not be the actual values used, but DrawSprocket will attempt to match your suggested size as closely as possible.

To find out what size dirty rectangle grid DrawSprocket is actually using, call `DSpContext_GetDirtyRectGridSize` (page 42). To find out the base grid size that all dirty rectangle grids must be a multiple of, use the function `DSpContext_GetDirtyRectGridUnits` (page 43).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetDirtyRectGridSize

Finds out the current grid size for a context's dirty rectangles.

```
OSStatus DSpContext_GetDirtyRectGridSize (
                        DSpContextReference inContext,
                        UInt32 *outCellPixelWidth,
                        UInt32 *outCellPixelHeight);
```

`inContext` A reference to a context for which you want to know the current grid cell size of the dirty rectangles.

`outCellPixelWidth`
On return, the width of the grid cell in pixels.

`outCellPixelHeight`
On return, the height of the grid cell in pixels.

function result A result code. See "Result Codes" (page 87).

DISCUSSION

The height and width values may be different from the values specified in `DSpContext_SetDirtyRectGridSize` because the grid cells must be multiples of the base grid size. For example, if you request a grid cell size of 40 by 40 pixels on the current PowerPC machines, the actual cell size will be 64 by 64 because the base grid size is 32 by 32 pixels. To find out the dimensions of the base grid, you can use the `DSpContext_GetDirtyRectGridUnits` (page 43) function.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetDirtyRectGridUnits

Finds out the size of the base dirty rectangle grid for a context.

```
OSStatus DSpContext_GetDirtyRectGridUnits (
                     DSpContextReference inContext,
                     UInt32 *outCellPixelWidth,
                     UInt32 *outCellPixelHeight);
```

`inContext`	A reference to a context for whose base dirty rectangle grid size you want to determine.
`outCellPixelWidth`	On return, the width of the base grid in pixels.
`outCellPixelHeight`	On return, the height of the base grid in pixels.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

The grid unit size is based on a number of machine characteristics such as the bus width and L1 cache size. For example, on current PowerPC-based machines, the grid unit size is 32 by 32 pixels (corresponding to the 32 bytes that make up the width of a PowerPC cache line). When you specify a grid cell size with the `DSpContext_SetDirtyRectGridSize` function, DrawSprocket rounds the requested size to a multiple of the base grid unit size. For example, if you request a grid cell size of 40 by 40 pixels, the actual cell size will be 64 by 64.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_SetMaxFrameRate

Sets a maximum frame rate for a specified context.

```
OSStatus DSpContext_SetMaxFrameRate (
                    DSpContextReference inContext,
                    UInt32 inMaxFPS);
```

`inContext`	A reference to the context whose maximum frame rate you want to set.
`inMaxFPS`	The maximum frame rate in frames per second.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

A call to the function `DSpContext_SetMaxFrameRate` does not guarantee that your game will achieve the maximum rate, but if it attempts to exceed the rate, DrawSprocket will slow down the buffer swapping.

The actual frame rate that is set is not necessarily the frame rate you specified, because DrawSprocket internally converts the specified maximum frame rate into a value that can be used to skip a number of frames for each frame that is drawn.

For example, if the monitor refresh rate is 66.7 Hz, and you request a frame rate of 30 fps, DrawSprocket internally skips every other frame, and your resulting frame rate is about 33.3 Hz.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetMaxFrameRate

Obtains the maximum frame rate for a specified context.

```
OSStatus DSpContext_GetMaxFrameRate (
                    DSpContextReference inContext,
                    UInt32 *outMaxFPS);
```

`inContext`	A reference to the context whose maximum frame rate you want to get.
`outMaxFPS`	On return, the maximum frame rate in frames per second for the context specified in the `inContext` parameter. The frame rate given is not necessarily the same as the maximum frame rate passed by the most recent call to the `DSpContext_SetMaxFrameRate` function. If 0 is given as the maximum frame rate, there are no frame rate restrictions in place.
function result	A result code. See "Result Codes" (page 87).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetMonitorFrequency

Obtains the frequency for the display associated with a context.

```
OSStatus DSpContext_GetMonitorFrequency (
                    DSpContextReference inContext,
                    Fixed *outFrequency);
```

`inContext`	A reference to a context for which you want to get the display frequency.

`outFrequency`	On return, the display frequency. The context must have been active for a reasonable amount of time (at least two seconds) in order to receive a correct value, because the value given by this parameter on return is calculated by timing the frame rate of the active context.
function result	A result code. See “Result Codes” (page 87).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Blitting Functions

This section describes functions you use to blit images between buffers. These functions are generalized in the sense that you can copy images between any two buffers that can be represented by a `CGrafPort` reference.

- `DSpBlit_Faster` (page 46) performs the specified blitting operation (including scaling).
- `DSpBlit_Fastest` (page 47) performs the specified blitting operation (without scaling).

DSpBlit_Faster

Performs the specified blitting operation (including scaling).

```
OSStatus DSpBlit_Faster (
                    DSpBlitInfoPtr inBlitInfo,
                    Boolean inAsyncFlag);
```

`inBlitInfo`	A pointer to a structure that specifies the blitting operation you want to perform. See `DSpBlitInfo` (page 68) for more information.
`inAsyncFlag`	If set to `true`, DrawSprocket attempts to perform the blitting operation asynchronously.
function result	A result code. See "Result Codes" (page 87).

DESCRIPTION

If you specify asynchronous blitting, you must specify a completion function in the `inBlitInfo` structure which will be called when DrawSprocket finishes the blitting operation.

VERSION NOTES

Introduced with DrawSprocket 1.1.

DSpBlit_Fastest

Performs the specified blitting operation (without scaling).

```
OSStatus DSpBlit_Faster (
                        DSpBlitInfoPtr inBlitInfo,
                        Boolean inAsyncFlag);
```

`inBlitInfo`	A pointer to a structure that specifies the blitting operation you want to perform. See `DSpBlitInfo` (page 68) for more information.
`inAsyncFlag`	If set to `true`, DrawSprocket attempts to perform the blitting operation asynchronously.
function result	A result code. See "Result Codes" (page 87).

DESCRIPTION

Unlike `DSpBlit_Faster` (page 46), `DSpBlit_Fastest` forgoes checking for special drawing cases (such as clipping) when copying between buffers.

If you specify asynchronous blitting, you must specify a completion function in the `inBlitInfo` structure which will be called when DrawSprocket finishes the blitting operation.

VERSION NOTES

Introduced with DrawSprocket 1.1.

Using Alternate Buffers

Use the functions in this section to create and draw into an alternate buffer and to designate an alternate buffer to serve as an underlay.

- `DSpAltBuffer_New` (page 48) creates an alternate buffer for an underlay or overlay.
- `DSpAltBuffer_Dispose` (page 49) disposes of an alternate buffer.
- `DSpAltBuffer_GetCGrafPtr` (page 50) obtains the drawing area for an alternate buffer.
- `DSpContext_SetUnderlayAltBuffer` (page 51) designates an alternate buffer to be used as the current underlay buffer for a context.
- `DSpContext_GetUnderlayAltBuffer` (page 52) obtains the current underlay associated with a context.
- `DSpAltBuffer_InvalRect` (page 52) invalidates a rectangle in an alternate buffer.

DSpAltBuffer_New

Creates an alternate buffer for an underlay.

```
OSStatus DSpAltBuffer_New (
                    DSpContextReference inContext,
                    Boolean inVRAMBuffer,
                    DSpAltBufferAttributes *inAttributes,
                    DSpAltBufferReference *outAltBuffer);
```

`inContext`	A reference to the context for which you want to create an alternate buffer.
`inVRAMBuffer`	A value of `true` requests that DrawSprocket create the buffer in VRAM if possible (it may be created in the current heap). A value of `false` means to create the buffer in the current heap.
`inAttributes`	A pointer to a structure specifying additional attributes of the alternate buffer. See `DSpAltBufferAttributes` (page 68) for more information. If you pass `NULL`, the alternate buffer has the same attributes as the specified context.
`outAltBuffer`	On return, a pointer to the alternate buffer reference.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

If you specify additional attributes in the `inAttributes` parameter, you cannot use the alternate buffer as an underlay buffer.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpAltBuffer_Dispose

Disposes of an alternate buffer.

```
OSStatus DSpAltBuffer_Dispose  (DSpAltBufferReference inAltBuffer);
```

`inAltBuffer`	A reference to the buffer to dispose.
function result	A result code. See "Result Codes" (page 87).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpAltBuffer_GetCGrafPtr

Obtains the drawing area for an alternate buffer.

```
OSStatus DSpAltBuffer_GetCGrafPtr (
                    DSpAltBufferReference inAltBuffer,
                    DSpBufferKind inBufferKind,
                    CGrafPtr *outCGrafPtr);
```

`inAltBuffer`	A reference to an alternate buffer.
`inBufferKind`	The kind of buffer. Currently the only supported buffer kind is `kDSpBufferKind_Normal`.
`outCGrafPtr`	On return, the graphics pointer associated with an alternate buffer.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

After the `DSpAltBuffer_GetCGrafPtr` function returns, you can use the pointer indicated in `outCGrafPtr` to draw into the alternate buffer. After drawing into the alternate buffer, you should invalidate the rectangles that you have worked in using the function `DSpAltBuffer_InvalRect` (page 52).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_SetUnderlayAltBuffer

Designates an alternate buffer to be used as the current underlay buffer for a context.

```
OSStatus DSpContext_SetUnderlayAltBuffer (
                        DSpContextReference inContext,
                        DSpAltBufferReference inNewUnderlay);
```

`inContext`	A reference to the context that uses the underlay.
`inNewUnderlay`	A reference to the alternate buffer that holds the underlay.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

Underlay buffers are used to "clean" a back buffer when `DSpContext_GetBackBuffer` is called. When a back buffer is retrieved and there is an underlay buffer, the invalid areas in the back buffer are restored from the underlay buffer. This is most useful in sprite games, or in games where the background is static (or changes infrequently).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetUnderlayAltBuffer

Obtains the current underlay associated with a context.

```
OSStatus DSpContext_GetUnderlayAltBuffer (
                    DSpContextReference inContext,
                    DSpAltBufferReference *outUnderlay);
```

`inContext`	A reference to the context whose underlay you want to get.
`outUnderlay`	On return, a reference to the alternate buffer that holds the underlay.
function result	A result code. See "Result Codes" (page 87).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpAltBuffer_InvalRect

Invalidates a rectangle in an alternate buffer.

```
OSStatus DSpAltBuffer_InvalRect (
                    DSpAltBufferReference inAltBuffer,
                    const Rect *inInvalidRect);
```

`inAltBuffer`	A reference to an alternate buffer.
`inInvalidRect`	A pointer to the rectangle to be invalidated.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

For example, you must invalidate areas of an underlay you have changed so that the changes are transferred to the back buffer on the next `DSpContext_SwapBuffers` (page 39) call.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Handling a Mouse

Because the coordinate system of a context may not correspond exactly to the global system coordinates, you must use the functions in this section to track the position of the mouse.

- `DSpFindContextFromPoint` (page 53) finds out which context contains a point given in global coordinates.
- `DSpGetMouse` (page 54) obtains the global coordinates of the mouse position.
- `DSpContext_GlobalToLocal` (page 55) translates a point in global coordinates into local coordinates for a context.
- `DSpContext_LocalToGlobal` (page 55) translates a point from a context's local coordinates into global coordinates.

DSpFindContextFromPoint

Finds out which context contains a point given in global coordinates.

```
OSStatus DSpFindContextFromPoint (
                    Point inGlobalPoint,
                    DSpContextReference *outContext);
```

`inGlobalPoint`
: A point in global coordinates.

`outContext`
: On return, a reference to the context that contains that point.

function result
: A result code. See “Result Codes” (page 87).

DISCUSSION

If the user moves the mouse, the game needs to know which context contains it so that the global coordinates can be properly translated into local coordinates for the context.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpGetMouse

Obtains the global coordinates of the mouse position.

```
OSStatus DSpGetMouse  (Point *outGlobalPoint);
```

`outGlobalPoint`
: On return, the global coordinates of the mouse position.

function result
: A result code. See “Result Codes” (page 87).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GlobalToLocal

Translates a point in global coordinates into local coordinates for a context.

```
OSStatus DSpContext_GlobalToLocal (
                  DSpContextReference inContext,
                  Point *ioPoint);
```

`inContext`	A reference to the context whose local coordinates you want to translate into.
`ioPoint`	Takes a point in global coordinates. On return, contains the point in local coordinates.
function result	A result code. See "Result Codes" (page 87).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_LocalToGlobal

Translates a point from a context's local coordinates into global coordinates.

```
OSStatus DSpContext_LocalToGlobal (
                  DSpContextReference inContext,
                  Point *ioPoint);
```

`inContext`	The context whose local coordinate system describes the point's coordinates.
`ioPoint`	Takes a point's local coordinates. On return, contains the point's global coordinates.
function result	A result code. See "Result Codes" (page 87).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Manipulating Color Lookup Tables

The functions in this section give you convenient access to the entries of a color lookup table.

- `DSpContext_SetCLUTEntries` (page 56) assigns one or more color entries to a color lookup table.
- `DSpContext_GetCLUTEntries` (page 57) Retrieves one or more color entries from a color lookup table.

DSpContext_SetCLUTEntries

Assigns one or more color entries to a color lookup table.

```
OSStatus DSpContext_SetCLUTEntries (
                    DSpContextReference inContext,
                    const ColorSpec *inEntries,
                    UInt16 inStartingEntry,
                    UInt16 inEntryCount);
```

`inContext`	The context whose color lookup table is to be modified.
`inEntries`	A pointer to an array of color specification records.
`inStartingEntry`	The (zero-based) index position in the color lookup table of the first entry to replace.
`inEntryCount`	The number of entries to replace.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

The `DSpContext_SetCLUTEntries` function allows you to change a range of entries in a color lookup table, for purposes such as color-table animation.

Because of video hardware limitations, the changes you make to a color table with this function may not take effect until the next vertical retrace. Nevertheless, this function attempts to execute asynchronously and return immediately, so your program can continue execution without having to wait for the changes to be made.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_GetCLUTEntries

Retrieves one or more color entries from a color lookup table.

```
OSStatus DSpContext_GetCLUTEntries (
                    DSpContextReference inContext,
                    ColorSpec *outEntries,
                    UInt16 inStartingEntry,
                    UInt16 inEntryCount);
```

`inContext`	The context whose color lookup table is to be accessed.
`outEntries`	On return, an array of color specification records that contain the retrieved table entries.
`inStartingEntry`	The (zero-based) index position in the color lookup table of the first entry to retrieve.
`inEntryCount`	The number of entries to retrieve.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

After you get the entries you can modify them and reassign them to the color table, for purposes such as color-table animation, with the function `DSpContext_SetCLUTEntries` (page 56).

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Processing System Events

The function in this section passes system events through to DrawSprocket.

- `DSpProcessEvent` (page 58) passes system events through to DrawSprocket so that it can correctly handle events it must know about.

DSpProcessEvent

Passes system events through to DrawSprocket so that it can correctly handle events it must know about.

```
OSStatus DSpProcessEvent (
                    EventRecord *inEvent,
                    Boolean *outEventWasProcessed);
```

`inEvent`	A pointer to the event to be passed to DrawSprocket.
`outEventWasProcessed`	On return, `true` if DrawSprocket processed the event; `false` if the event was not processed.
function result	A result code. See “Result Codes” (page 87).

DISCUSSION

Whenever your game receives a suspend or resume event, it must call the `DSpProcessEvent` function so that DrawSprocket can correctly set the system state for the process switch.

When DrawSprocket is suspended, it returns the display to the resolution mode it was in before your context's play state first became active. When DrawSprocket resumes, it restores the display to the resolution mode used by your context. However, it is your responsibility to update the contents of the display at this time.

SPECIAL CONSIDERATIONS

Do not call at interrupt time.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Utility Functions

This section describes two functions: one that aids your debugging efforts by maintaining your access to the debugging screen at all times, and one that facilitates implementation of VBL tasks.

- `DSpSetDebugMode` (page 59) keeps the screen and system resources visible during debugging.
- `DSpContext_SetVBLProc` (page 60) piggybacks your own VBL task to a particular context.

DSpSetDebugMode

Keeps the screen and system resources visible during debugging.

```
OSStatus DSpSetDebugMode (Boolean inDebugMode);
```

`inDebugMode`	Set this value to `true` if the desktop display is to remain visible, even after fading; `false` otherwise.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

During development, if you drop into the debugger when the display has been faded out, you cannot fade the display back in so that you can see the debugger screen. Calling the `DSpSetDebugMode` function with the `inDebugMode` flag set to a value of `true` causes your program to enter a mode in which the blanking window is not drawn and every fade operation (either in or out) causes only a partial dimming and immediate restoration of the screen intensity. Calling this function with the `inDebugMode` flag set to a value of `false` ends the mode and resumes normal operation.

To make use of this function, you must call it before activating your context. Once the blanking window is in place, this function effects only gamma fades.

This function is ignored in nondebugging builds of DrawSprocket.

Note the when using debugging builds, you can also enter debug mode by placing a folder named `DSpDebugMode` in your application folder.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContext_SetVBLProc

Piggybacks your own VBL task to a particular context.

```
OSStatus DSpContext_SetVBLProc (
                    DSpContextReference inContext,
                    DSpCallbackProcPtr inProcPtr,
                    void *inRefCon);
```

`inContext`	The context the VBL task is associated with.

`inProcPtr`	A pointer to an application-supplied callback function. See `MyCallbackFunction` (page 62) and the data type `DSpCallbackProcPtr` (page 70) for more information about implementing this function.
`inRefCon`	A reference constant to be handed back by DrawSprocket when it calls the `inProcPtr` callback.
function result	A result code. See "Result Codes" (page 87).

DISCUSSION

Because DrawSprocket needs to set up VBL tasks of its own, you can piggyback your own VBL task to a particular context easily with this function, instead of digging down through the system to find the correct slot ID and installing your own.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Application-Defined Functions

This section describes the interfaces to application-defined functions used in conjunction with DrawSprocket functions.

- `MyCallbackFunction` (page 62) performs any necessary tasks in preparation for swapping display buffers or piggybacking VBL tasks to a context.
- `MyBlitDone` (page 62) handles any tasks required after DrawSprocket finishes blitting between buffers.
- `MyEventHandler` (page 63) allows your game to handle events during the display of the device-selection dialog box.

MyCallbackFunction

Performs any necessary tasks in preparation for swapping display buffers or piggybacking VBL tasks to a context.

```
Boolean MyCallbackFunction (
                     DSpContextReference inContext,
                     void *inRefCon);
```

`inContext`	A reference to a context.
`inRefCon`	A reference constant to be handed back to the game by the DrawSprocket function that calls `MyCallbackFunction`.
function result	The function should return `false` if your tasks or checks are complete. If it returns `true`, the function is still performing necessary tasks.

DISCUSSION

Calls to `MyCallbackFunction` result from calls to either `DSpContext_SetVBLProc` (page 60) or `DSpContext_SwapBuffers` (page 39).

VERSION NOTES

Introduced with DrawSprocket 1.0.

MyBlitDone

Handles any tasks required after DrawSprocket finishes blitting between buffers.

```
void MyBlitDone (DSpBlitInfo *info);
```

`info`	A pointer to a data structure containing information about the completed blitting operation. See `DSpBlitInfo` (page 68) for more information.

DISCUSSION

DrawSprocket calls this application-defined function during calls to the functions `DSpBlit_Faster` (page 46) or `DSpBlit_Fastest` (page 47).

If you are performing multiple asynchronous blitting operations, your application-defined completion function can check the blitter information structure passed to it to determine which operation was completed.

VERSION NOTES

Introduced with DrawSprocket 1.1

MyEventHandler

Handles events during calls to the function `DSpUserSelectContext` (page 20).

```
Boolean MyEventHandler (EventRecord *inEvent);
```

`inEvent`	A pointer to an event record that describes the event that occurred.
function result	If your function handled the event, it should return `true`; otherwise it should return `false`.

DISCUSSION

When calling the function `DSpUserSelectContext` (page 20), you must designate this application-defined function to handle events (such as update events) that may occur while the configuration window is active.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Data Types

This section describes the following DrawSprocket data types:

- `DSpContextReference` (page 64)
- `DSpAltBufferReference` (page 64)
- `DSpContextAttributes` (page 65)
- `DSpAltBufferAttributes` (page 68)
- `DSpBlitInfo` (page 68)
- `DSpCallbackProcPtr` (page 70)
- `DSpBlitDoneProc` (page 71)
- `DSpEventProcPtr` (page 71)

DSpContextReference

DrawSprocket handles drawing contexts by passing a reference of type `DSpContextReference`:

```
typedef struct OpaqueDSpContextReference    *DSpContextReference;
```

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpAltBufferReference

DrawSprocket handles the alternate image buffer by passing a reference of type `DSpAltBufferReference`:

```
typedef struct OpaqueDSpAltBufferReference  *DSpAltBufferReference;
```

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpContextAttributes

The context attributes structure describes a set of characteristics that apply to a given context.

You use the context attributes structure to request specific characteristics when creating a context or to retrieve the actual characteristics of a given context. The field descriptions cover their use as both input or output values, but the structure never contains both input and output information at the same time. The context attributes structure is defined by the `DSpContextAttributes` data type.

Note
You can use the debug version of the DrawSprocket library to catch most context errors. ◆

```
struct DSpContextAttributes {
    Fixed               frequency;
    UInt32              displayWidth;
    UInt32              displayHeight;
    UInt32              reserved1;
    UInt32              reserved2;
    UInt32              colorNeeds;
    CTabHandle          colorTable;
    OptionBits          contextOptions;
    OptionBits          backBufferDepthMask;
    OptionBits          displayDepthMask;
    UInt32              backBufferBestDepth;
    UInt32              displayBestDepth;
    UInt32              pageCount;
    char                filler[3],
    Boolean             gameMustConfirmSwitch;
    UInt32              reserved3[4];
};
```

```
typedef struct DSpContextAttributes DSpContextAttributes;
typedef struct DSpContextAttributes *DSpContextAttributesPtr
```

Field descriptions

`frequency`	*Input*: Ignored. *Output*: The frame-refresh frequency (in Hz) specified by the current resolution mode. (This value is 0 if the actual frequency is not available.)
`displayWidth`	*Input:* The requested display width (in pixels). *Output*: The display width for the specified context.
`displayHeight`	*Input:* The requested display height (in pixels). *Output:* The display height for the specified context.
`reserved1`	Reserved. Always set this field to 0.
`reserved2`	Reserved. Always set this field to 0
`colorNeeds`	*Input*: A value that specifies whether the display needs to be in color. Valid constants for this field are described in “Color Need Constants” (page 73). *Output*: The color support provided by the current resolution mode.
`colorTable`	*Input:* A handle to the color table to use with the context to which this attributes structure applies. (This field applies only to indexed devices; direct devices do not use a color table.) *Output*: Ignored.
`contextOptions`	*Input*: A set of bit flags that define requested special display features for which either hardware or software implementation is acceptable. Valid constants for this field are described in “Special Display Feature Constants” (page 73). *Output*: The special display features supported in software by the current resolution mode.
`backBufferDepthMask`	*Input:* A bit array that defines the acceptable pixel depths for the back buffer. Valid constants for this field are described in “Depth Masks” (page 72). This value should match the depth of the front buffer. You must specify a back buffer depth mask and pixel depth when reserving a back buffer context.

	Output: The bit depth DrawSprocket recommends for the context.
`displayDepthMask`	*Input:* A bit array that defines the acceptable pixel depths for the front buffer. Valid constants for this field are described in “Depth Masks” (page 72). *Output:* A bit array that specifies the pixel depth of the specified context.
`backBufferBestDepth`	*Input:* The preferred pixel depth, or video mode, for the back buffer. his value should match the depth of the front buffer. You must specify a back buffer depth mask and pixel depth when reserving a back buffer context. *Output*: The bit depth DrawSprocket recommends for the context.
`displayBestDepth`	*Input:* The preferred pixel depth for the display. *Output:* The pixel depth of the specified context.
`pageCount`	*Input*: Indicates the desired number of video pages. For example, if you desire double-buffering, you should pass 2. For triple buffering, pass 3. If you pass 1, then DrawSprocket only provides a front buffer and does not allocate any memory for back buffers. You cannot pass 0 for the page count. *Output*: Gives the number of hardware video pages available. A value of 1 indicates that hardware page flipping is not supported.
`filler`	Reserved. These bytes are included to preserve alignment.
`gameMustConfirmSwitch`	*Input*: Ignored. *Output*: A value of `true` indicates that the context may have problems being displayed on the user’s system, and the game should confirm that the context is visible after being set to the active state by asking the user if the display can be seen (via a dialog box or some other mechanism). Additionally, a warning code will be returned from `DSpContext_SetState` indicating that the game should confirm that the context is visible.
`reserved3[4]`	Reserved. Always set this field to 0.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpAltBufferAttributes

When handling allocating an alternate drawing buffer, you can specify additional attributes by passing a structure of type `DSpAltBufferAttributes`.

```
struct DSpAltBufferAttributes {
    UInt32 width;
    UInt32 height;
    DSpAltBufferOption options;
    UInt32 reserved[4];
};
typedef struct DSpAltBufferAttributes DSpAltBufferAttributes;
```

Field descriptions

`width`	The width of the alternate buffer, in pixels.
`height`	The height of the alternate buffer, in pixels.
`options`	Any desired options for the alternate buffer. See "Alternate Buffer Options Constant" (page 76).
`reserved[4]`	Reserved. Set to 0.

VERSION NOTES

Introduced with DrawSprocket 1.1.

DSpBlitInfo

When blitting images between buffers, you specify the type of blitting operation by passing a structure of type `DSpBlitInfo`.

```
struct DSpBlitInfo {
    Boolean completionFlag;
    char filler[3];
    DSpBlitDoneProc completionProc;
    DSpContextReference srcContext;
    CGrafPtr srcBuffer;
    Rect srcRect;
    UInt32 srcKey;
    DSpContextReference dstContext;
    CGrafPtr dstBuffer;
    Rect dstRect;
    UInt32 dstKey;
    DSpBlitMode mode;
    UInt32 reserved[4];
};
typedef struct DSpBlitInfo DSpBlitInfo;
typedef DSpBlitInfo DSpBlitInfoPtr;
```

Field descriptions

`completionFlag`	Set to `true` on output when the blitting operation has completed.
`filler[3]`	Reserved. These bytes are included to preserve proper alignment.
`completionProc`	A pointer to the function that DrawSprocket should call when the blitting operation has completed. See `MyBlitDone` (page 62) for more information about implementing this function. Pass `NULL` if you don't want to specify a completion function.
`srcContext`	A reference to the source context. Pass `NULL` if the source buffer does not belong to a context.
`srcBuffer`	A pointer of type `CGrafPtr` that specifes the buffer containing the image data you want to blit to the destination buffer.
`srcRect`	The rectangle specifying the location of the image data in the source buffer. If the source and destination rectangles are different sizes, DrawSprocket scales the image to fit.
`srcKey`	An integer specifying the source color key. See "Blit Mode Constants" (page 77) for more information on using this key.
`dstContext`	A reference to the destination context. Pass `NULL` if the destination buffer does not belong to a context.

`dstBuffer`	A pointer of type `CGrafPtr` that specifes the buffer you want to blit the image to.
`dstRect`	The rectangle specifying where to write the image data in the destination buffer. If the source and destination rectangles are different sizes, DrawSprocket scales the image to fit.
`dstKey`	An integer specifying the destination color key. See "Blit Mode Constants" (page 77) for more information on using this key.
`mode`	The blit mode to use. See "Blit Mode Constants" (page 77) for a list of possible values.
`reserved[4]`	Reserved.

VERSION NOTES

Introduced with DrawSprocket 1.1

DSpCallbackProcPtr

When calling the function `DSpContext_SetVBLProc` (page 60) or `DSpContext_SwapBuffers` (page 39), you must designate an application-defined function to handle any desired operations prior to the VBL task call or buffer swap. Such a function has the following type definition:

```
typedef Boolean (*DSpCallbackProcPtr)(DSpContextReference inContext,
                  void *inRefCon);
```

See `MyCallbackFunction` (page 62) for more information about implementing this function.

VERSION NOTES

Introduced with DrawSprocket 1.0.

DSpBlitDoneProc

When passing the DSpBlitInfo structure in the functions `DSpBlit_Faster` (page 46) and `DSpBlit_Fastest` (page 47), you must designate a completion function to be called when DrawSprocket finishes blitting to the screen. Such a function has the following type definition:

```
typedef void (*DSpBlitDoneProc)(DSpBlitInfo *info);
```

See `MyBlitDone` (page 62) for more information about implementing this function.

VERSION NOTES

Introduced with DrawSprocket 1.1

DSpEventProcPtr

When calling the function `DSpUserSelectContext` (page 20), you must designate an application-defined function to handle events. Such a function has the following type definition:

```
typedef Boolean (*DSpEventProcPtr)(EventRecord *inEvent);
```

See `MyEventHandler` (page 63) for more information about implementing this function.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Constants

This section describes the constants provided by DrawSprocket.

Depth Masks

You provide a depth mask in the `backBufferDepthMask` and `displayDepthMask` fields of the context attributes structure to specify the pixel depths that are acceptable to your program. You can construct the mask from the constants defined by the following enumeration:

```
enum DSpDepthMask {
    kDSpDepthMask_1                = 1 << 0,
    kDSpDepthMask_2                = 1 << 1,
    kDSpDepthMask_4                = 1 << 2,
    kDSpDepthMask_8                = 1 << 3,
    kDSpDepthMask_16               = 1 << 4,
    kDSpDepthMask_32               = 1 << 5,
    kDSpDepthMask_All              = -1L
};

typedef enum DSpDepthMask DSpDepthMask;
```

Constant descriptions

`kDSpDepthMask_1`	1-bit pixel depth is acceptable.
`kDSpDepthMask_2`	2-bit pixel depth is acceptable.
`kDSpDepthMask_4`	4-bit pixel depth is acceptable.
`kDSpDepthMask_8`	8-bit pixel depth is acceptable.
`kDSpDepthMask_16`	16-bit pixel depth is acceptable.
`kDSpDepthMask_32`	32-bit pixel depth is acceptable.

`kDSpDepthMask_All` Any pixel depth is acceptable.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Color Need Constants

You can use the following constants in the `colorNeeds` field of the context attributes structure to specify your program's preferences or requirements for color display:

```
enum DSpColorNeeds {
    kDSpColorNeeds_DontCare         = 0L,
    kDSpColorNeeds_Request          = 1L,
    kDSpColorNeeds_Require          = 2L
};

typedef enum DSpColorNeeds DSpColorNeeds;
```

Constant descriptions

`kDSpColorNeeds_DontCare`
Display can be either color or grayscale.

`kDSpColorNeeds_Request`
Color display is preferred, but not required.

`kDSpColorNeeds_Require`
Color display is required.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Special Display Feature Constants

You can use the following constants in the `contextOptions` field of the context attributes structure to request special display features or to determine which features a specified display supports.

```
enum DSpContextOption {
    kDSpContextOption_QD3DAccel     = 1 << 0,
    kDSpContextOption_PageFlip      = 1 << 1,
    kDSpContextOption_DontSyncVBL   = 1 << 2
};

typedef enum DSpContextOption DSpContextOption;
```

Constant descriptions

`kDSpContextOption_QD3DAccel`
: Not implemented.

`kDSpContextOption_PageFlip`
: Use page flipping (a hardware feature). Note that you should never allow page flipping unless you have tested your code extensively on a computer with page-flipping capability.

`kDSpContextOption_DontSyncVBL`
: Do not synchronize context updates with the vertical retrace of the display.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Buffer Kind Constant

Currently, DrawSprocket supports only one kind of buffer. You pass this constant to the `DSpContext_GetBackBuffer`, `DSpAltBuffer_InvalRect`, and `DSpAltBuffer_GetCGrafPtr` functions.

```
enum DSpBufferKind {
    kDSpBufferKind_Normal           = 0
};

typedef enum DSpBufferKind DSpBufferKind;
```

VERSION NOTES

Introduced with DrawSprocket 1.0.

Play State Constants

You set the play state of a context by calling the function `DSpContext_SetState` (page 28) and passing it one of the following values:

```
enum DSpContextState {
    kDSpContextState_Active          = 0L,
    kDSpContextState_Paused          = 1L,
    kDSpContextState_Inactive        = 2L
};

typedef enum DSpContextState DSpContextState;
```

Constant Descriptions

`kDSpContextState_Active`
: The display is completely controlled by your program. The display is configured as specified in its context attributes structure. All system adornments, such as the menu bar, floating windows, and the desktop, are hidden (removed or covered by the blanking window). In this state you cannot make system calls to managers, such as the Window Manager and Dialog Manger, that expect the display to be in a normal state.

`kDSpContextState_Paused`
: The menu bar and other system adornments are restored, although the resolution mode is still that specified in the context attributes structure for the display. The desktop is still not visible; the blanking window covers it. In this state you can make normal system calls. Page flipping and double buffering are inactive in this state; the display page is set to page 0 if page flipping has been enabled.

  In this state it is safe for your program to call Macintosh Toolbox and system software functions. The paused state gives the user access to the process menu; if your game is suspended because the user switches to another application, you must call the `DSpProcessEvent` function (page 58).

`kDSpContextState_Inactive`
: The display is in exactly the state the user has specified from the Monitors control panel. The blanking window is

hidden and the resolution mode specified in the context attributes structure for this display is not in effect.

The user's configuration is restored only if there are no other currently active or paused contexts. As long as there is at least one active or paused context, all displays are covered by the blanking window, and the resolution mode for each is that of the context, which may not be what the user has selected in the Monitors control panel.

VERSION NOTES

Introduced with DrawSprocket 1.0.

Alternate Buffer Options Constant

You can specify this constant when allocating an alternate display buffer.

```
enum DSpAltBufferOption {
    kDSpAltBufferOption_RowBytesEqualsWidth = 1 << 0
    };
typedef enum DSpAltBufferOption DSpAltBufferOption;
```

Constant description

`kDSpAltBufferOption_RowBytesEqualsWidth`

Forces the row and width of the alternate buffer to have the same number of pixels. The number of row bytes can vary depending on the screen depth. For example, if you specify 16-bit color, then there will be twice as many row bytes as there are pixels in the width, because it takes 2 bytes to represent one pixel.

VERSION NOTES

Introduced with DrawSprocket 1.1.

Blit Mode Constants

You use these constants in the structure `DSpBlitInfo` (page 68) to indicate the type of blitter operation you want to perform. Note that you can use these constants in combination with each other.

```
enum DSpBlitMode {
    kDSpBlitMode_Plain           = 0,
    kDSpBlitMode_SrcKey          = 1 << 0,
    kDSpBlitMode_DstKey          = 1 << 1,
    kDSpBlitMode_Interpolation   = 1 << 2
    };
typedef enum DSpBlitMode DSpBlitMode;
```

Constant descriptions

`kDSpBlitMode_Plain`
: Copy all pixels from the source to the destination.

`kDSpBlitMode_SrcKey`
: Copies all image data where the source image is *not* the same color as the source key. For example, say the buffer holds a sprite image on a black background. If you specify the source color key to be black, then DrawSprocket writes only nonblack images (that is, the sprite) to the destination.

`kDSpBlitMode_DstKey`
: Overwrites data in the destination image where the color is the same as the destination key. For example, say the destination buffer holds an image of a a city skyline against a blue sky, and you want to draw a blimp moving behind the buildings. If you set the destination color key to blue, then DrawSprocket will draw the blimp only in areas that are blue. That is, the blimp will not overwrite the nonblue buildings, so it will appear to be behind them.

`kDSpBlitMode_Interpolation`
: Interpolate between color values when scaling pixels.

VERSION NOTES

Introduced with DrawSprocket 1.1.

Every Context Constant

DrawSprocket defines the following constant that you can pass to indicate that any context is permissible:

```
#define kDSpEveryContext ((DSpContextReference) NULL)
```

VERSION NOTES

Introduced with DrawSprocket 1.0.

Summary of DrawSprocket

DrawSprocket Functions

Activating and Deactivating DrawSprocket

```
NumVersion DSpGetVersion            (void);

OSStatus DSpStartup                 (void);

OSStatus DSpShutdown                (void);
```

Choosing a Context

```
OSStatus DSpFindBestContext         (const DSpContextAttributesPtr inDesiredAttributes,
                                     DSpContextReference *outContext);

OSStatus DSpFindBestContextOnDisplayID  (DSpContextAttributesPtr  inDesiredAttributes,
                                         DSpContextReference *outContext,
                                         DisplayIDType inDisplayID);

OSStatus DSpContext_GetDisplayID                (DSpContextReference inContext,
                                                 DisplayIDType *outDisplayID);

OSStatus DSpGetFirstContext         (DisplayIDType displayID,
                                     DSpContextReference *outContext);

OSStatus DSpGetNextContext          (DSpContextReference inCurrentContext,
                                     DSpContextReference *outContext);

OSStatus DSpContext_GetAttributes   (DSpContextReference inContext,
                                     DSpContextAttributesPtr outAttributes);

OSStatus DSpCanUserSelectContext    (DSpContextAttributesPtr inDesiredAttributes
                                     Boolean *outUserCanSelectContext );
```

```
OSStatus DSpUserSelectContext      (DSpContextAttributesPtr inDesiredAttributes,
                                    DisplayIDType inDialogDisplayLocation,
                                    DSpEventProcPtr inEventProc,
                                    DSpContextReference *outContext );
```

Saving and Restoring a Context

```
OSStatus DSpContext_Restore        (void *inFlatContext,
                                    DSpContextReference *outRestoredContext);

OSStatus DSpContext_GetFlattenedSize          (DSpContextReference inContext,
                                               UInt32 *outFlatContextSize);

OSStatus DSpContext_Flatten                   (DSpContextReference inContext,
                                               void *outFlatContext);
```

Manipulating a Context

```
OSStatus DSpContext_Reserve    (DSpContextReference inContext,
                                const DSpContextAttributesPtr inDesiredAttributes);

OSStatus DSpContext_Queue      (DSpContextReference inParentContext,
                                DSpContextReference inChildContext,
                                DSpContextAttributesPtr inDesiredAttributes);

OSStatus DSpContext_Switch     (DSpContextReference inOldContext,
                                DSpContextReference inNewContext);

OSStatus DSpContext_Release                   (DSpContextReference inContext);

OSStatus DSpContext_SetState                  (DSpContextReference inContext,
                                               DSpContextState inState);

OSStatus DSpContext_GetState                  (DSpContextReference inContext,
                                               DSpContextState *outState);

OSStatus DSpSetBlankingColor                  (const RGBColor *inRGBColor);
```

Drawing and Double Buffering

```
OSStatus DSpContext_FadeGamma                 (DSpContextReference inContext,
                                               SInt32 inPercentOfOriginalIntensity,
                                               RGBColor *inZeroIntensityColor);
```

```
OSStatus DSpContext_FadeGammaOut             (DSpContextReference inContext,
                                              RGBColor *inZeroIntensityColor);

OSStatus DSpContext_FadeGammaIn              (DSpContextReference inContext,
                                              RGBColor *inZeroIntensityColor);

OSStatus DSpContext_GetFrontBuffer           (DSpContextReference inContext,
                                              CGrafPtr *outBackBuffer);

OSStatus DSpContext_GetBackBuffer            (DSpContextReference inContext,
                                              DSpBufferKind inBufferKind,
                                              CGrafPtr *outBackBuffer);

OSStatus DSpContext_InvalBackBufferRect      (DSpContextReference inContext,
                                              const Rect *inRect);

OSStatus DSpContext_SwapBuffers              (DSpContextReference inContext,
                                              DSpCallbackProcPtr inBusyProc,
                                              void *inUserRefCon);

OSStatus DSpContext_IsBusy                   (DSpContextReference inContext,
                                              Boolean *outBusyFlag);

OSStatus DSpContext_SetDirtyRectGridSize     (DSpContextReference inContext,
                                              UInt32 inCellPixelWidth,
                                              UInt32 inCellPixelHeight);

OSStatus DSpContext_GetDirtyRectGridSize     (DSpContextReference inContext,
                                              UInt32 *outCellPixelWidth,
                                              UInt32 *outCellPixelHeight);

OSStatus DSpContext_GetDirtyRectGridUnits    (DSpContextReference inContext,
                                              UInt32 *outCellPixelWidth,
                                              UInt32 *outCellPixelHeight);

OSStatus DSpContext_SetMaxFrameRate          (DSpContextReference inContext,
                                              UInt32 inMaxFPS);

OSStatus DSpContext_GetMaxFrameRate          (DSpContextReference inContext,
                                              UInt32 *outMaxFPS);

OSStatus DSpContext_GetMonitorFrequency      (DSpContextReference inContext,
                                              Fixed *outFrequency);
```

Blitting Functions

```
OSStatus DSpBlit_Faster                      (DSpBlitInfoPtr inBlitInfo,
                                              Boolean inAsyncFlag);

OSStatus DSpBlit_Fastest                     (DSpBlitInfoPtr inBlitInfo,
                                              Boolean inAsyncFlag);
```

Using Alternate Buffers

```
OSStatus DSpAltBuffer_New                    (DSpContextReference inContext,
                                              Boolean inVRAMBuffer,
                                              DSpAltBufferAttributes *inAttributes,
                                              DSpAltBufferReference *outAltBuffer);

OSStatus DSpAltBuffer_Dispose                (DSpAltBufferReference inAltBuffer);

OSStatus DSpAltBuffer_GetCGrafPtr            (DSpAltBufferReference inAltBuffer,
                                              DSpBufferKind inBufferKind,
                                              CGrafPtr *outCGrafPtr);

OSStatus DSpContext_SetUnderlayAltBuffer     (DSpContextReference inContext,
                                              DSpAltBufferReference inNewUnderlay);

OSStatus DSpContext_GetUnderlayAltBuffer     (DSpContextReference inContext,
                                              DSpAltBufferReference *outUnderlay);

OSStatus DSpAltBuffer_InvalRect              (DSpAltBufferReference inAltBuffer,
                                              const Rect *inInvalidRect);
```

Handling a Mouse

```
OSStatus DSpFindContextFromPoint             (Point inGlobalPoint,
                                              DSpContextReference *outContext);

OSStatus DSpGetMouse                         (Point *outGlobalPoint);

OSStatus DSpContext_GlobalToLocal            (DSpContextReference inContext,
                                              Point *ioPoint);

OSStatus DSpContext_LocalToGlobal            (DSpContextReference inContext,
                                              Point *ioPoint);
```

Manipulating Color Lookup Tables

```
OSStatus DSpContext_SetCLUTEntries          (DSpContextReference inContext,
                                             const ColorSpec *inEntries,
                                             UInt16 inStartingEntry,
                                             UInt16 inEntryCount);

OSStatus DSpContext_GetCLUTEntries          (DSpContextReference inContext,
                                             ColorSpec *outEntries,
                                             UInt16 inStartingEntry,
                                             UInt16 inEntryCount);
```

Processing System Events

```
OSStatus DSpProcessEvent                    (EventRecord *inEvent,
                                             Boolean *outEventWasProcessed);
```

Utility Functions

```
OSStatus DSpSetDebugMode                    (Boolean inDebugMode);

OSStatus DSpContext_SetVBLProc              (DSpContextReference inContext,
                                             DSpCallbackProcPtr inProcPtr,
                                             void *inRefCon);
```

Application-Defined Functions

```
Boolean MyCallbackFunction                  (DSpContextReference inContext,
                                             void *inRefCon);

void MyBlitDone                             (DSpBlitInfo *info);

Boolean MyEventHandler                      (EventRecord* inEvent);
```

Data Types

```
typedef struct OpaqueDSpContextReference    *DSpContextReference;
typedef struct OpaqueDSpAltBufferReference  *DSpAltBufferReference;

typedef Boolean (*DSpEventProcPtr)          (EventRecord *inEvent);

typedef Boolean (*DSpCallbackProcPtr)       (DSpContextReference inContext,
                                             void *inRefCon);

typedef void (*DSpBlitDoneProc)             (DSpBlitInfo *info);
```

Context Attributes Structure

```
struct DSpContextAttributes {
    Fixed               frequency;
    UInt32              displayWidth;
    UInt32              displayHeight;
    UInt32              reserved1;
    UInt32              reserved2;
    UInt32              colorNeeds;
    CTabHandle          colorTable;
    OptionBits          contextOptions;
    OptionBits          backBufferDepthMask;
    OptionBits          displayDepthMask;
    UInt32              backBufferBestDepth;
    UInt32              displayBestDepth;
    UInt32              pageCount;
    char                filler[3],
    Boolean             gameMustConfirmSwitch;
    UInt32              reserved3[4];
}

typedef struct DSpContextAttributes DSpContextAttributes;
typedef struct DSpContextAttributes *DSpContextAttributesPtr
```

Alternate Drawing Buffer Attributes

```
struct DSpAltBufferAttributes {
    UInt32 width;
    UInt32 height;
    DSpAltBufferOption options;
    UInt32 reserved[4];
};
typedef struct DSpAltBufferAttributes DSpAltBufferAttributes;
```

Blitting Information Structure

```
struct DSpBlitInfo {
    Boolean completionFlag;
    char filler[3];
    DSpBlitDoneProc completionProc;
    DSpContextReference srcContext;
    CGrafPtr srcBuffer;
    Rect srcRect;
    UInt32 srcKey;
    DSpContextReference dstContext;
    CGrafPtr dstBuffer;
    Rect dstRect;
    UInt32 dstKey;
    DSpBlitMode mode;
    UInt32 reserved[4];
};
typedef struct DSpBlitInfo DSpBlitInfo;
```

Constants

Depth Masks

```
enum DSpDepthMask {
    kDSpDepthMask_1                = 1U<<0,
    kDSpDepthMask_2                = 1U<<1,
    kDSpDepthMask_4                = 1U<<2,
    kDSpDepthMask_8                = 1U<<3,
    kDSpDepthMask_16               = 1U<<4,
    kDSpDepthMask_32               = 1U<<5,
    kDSpDepthMask_All              = ~0U
};

typedef enum DSpDepthMask DSpDepthMask;
```

Color Need Constants

```
enum DSpColorNeeds {
    kDSpColorNeeds_DontCare         = 0L,
    kDSpColorNeeds_Request          = 1L,
    kDSpColorNeeds_Require          = 2L
};

typedef enum DSpColorNeeds DSpColorNeeds;
```

Special Display Features

```
enum DSpContextOption {
    kDSpContextOption_QD3DAccel     = 1 << 0,
    kDSpContextOption_PageFlip      = 1 << 1,
    kDSpContextOption_DontSyncVBL   = 1 << 2
};

typedef enum DSpContextOption DSpContextOption;
```

Buffer Kind

```
enum DSpBufferKind {
    kDSpBufferKind_Normal           = 0
};

typedef enum DSpBufferKind DSpBufferKind;
```

Play State

```
enum DSpContextState {
    kDSpContextState_Active         = 0L,
    kDSpContextState_Paused         = 1L,
    kDSpContextState_Inactive       = 2L
};

typedef enum DSpContextState DSpContextState;
```

Alternate Buffer Option Constant

```
enum DSpAltBufferOption {
    kDSpAltBufferOption_RowBytesEqualsWidth = 1 << 0
    };
typedef enum DSpAltBufferOption DSpAltBufferOption;
```

Blit Mode Constants

```
enum DSpBlitMode {
    kDSpBlitMode_Plain           = 0,
    kDSpBlitMode_SrcKey          = 1 << 0,
    kDSpBlitMode_DstKey          = 1 << 1,
    kDSpBlitMode_Interpolation   = 1 << 2
    };
typedef enum DSpBlitMode DSpBlitMode;
```

Every Context Constant

```
#define kDSpEveryContext ((DSpContextReference) NULL)
```

Result Codes

`kDSpNotInitializedErr`	–30440L	`DSpStartup` has not yet been called.
`kDSpSystemSWTooOldErr`	–30441L	System software too old.
`kDSpInvalidContextErr`	–30442L	Invalid context reference.
`kDSpInvalidAttributesErr`	–30443L	Some field in an attributes structure has an invalid value.
`kDSpContextAlreadyReservedErr`	–30444L	The context is already reserved.
`kDSpContextNotReservedErr`	–30445L	The context is not reserved.
`kDSpContextNotFoundErr`	–30446L	DrawSprocket couldn't find the context.
`kDSpFrameRateNotReadyErr`	–30447L	Not enough time has passed for DrawSprocket to calculate a frame rate.
`kDSpConfirmSwitchWarning`	–30448L	The `gameMustConfirmSwitch` flag is set.
`kDSpInternalErr`	–30449L	Corrupted DrawSprocket or other error.
`kDSpStereoContextErr`	-30450L	DrawSprocket attempted to process a stereo context. (DrawSprocket no longer supports GoggleSprocket.)

Document Version History

This document has had the following releases:

Table A-1 DrawSprocket documentation revision history

Version	**Notes**
October 20, 1999	First preliminary release. This document reflects the changes to DrawsSprocket since version 1.0 documented in Chapter 2 of the Apple Game Sprockets Guide. A summary of changes is as follows: New functions added: `DSpGetVersion` (page 10), `DSpGetCurrentContext` (page 15),`DSpFindBestContextOnDisplayID` (page 14), `DSpContext_Reserve` (page 25), `DSpContextQueue` (page 26), `DSpContextSwitch` (page 27), `DSpContext_GetFrontBuffer` (page 36), `DSpBlit_Faster` (page 46), `DSpBlit_Fastest` (page 47), and `MyBlitDone` (page 62). The function `DSpAltBuffer_New` (page 48) now takes an additional parameter: a pointer to an alternate buffer attributes structure, `DSpAltBufferAttributes` (page 68). Note that DrawSprocket still supports the older version of this call. DrawSprocket no longer supports GoggleSprocket. This document does not contain any GoggleSprocket-related constants (such as display features and buffer kind). Attempts to invoke GoggleSprocket-related features will return the new result code `kDSpStereoContextErr`. The context option constant `kDSpContextOption_TripleBuffering` replaced by `kDSpContextOption_DontUseVBL`. You now specify triple buffering by passing 3 in the `pageCount` parameter of the `DSpContextAttributes` structure. Pixel scaling and overlays were documented but never actually implemented; this document does not contain any APIs related to scaling or overlays. That is, the following functions no longer appear: `DSpContext_SetScale`, `DSpContext_GetScale`, `DSpContext_SetOverlayAltBuffer`, `DSpContext_GetOverlayAltBuffer`, and `DSpAltBuffer_RebuildTransparencyMask`. The pixel scaling constants were also removed.

Table A-1 DrawSprocket documentation revision history

Version	Notes
	Application-defined function `MyEventHandler` (page 63) no longer requires Pascal calling conventions.

Index

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER
Jun Suzuki

Special thanks to Chris DeSalvo, Geoff Stahl, George Warner, Tim Carroll, and Jasjeet Thind.

Acknowledgements to Dave Bice, Judy Helfland, Tim Monroe, and Larry Wood, who wrote the previous Game Sprockets guide.