FxPlug SDK Overview

Apple Applications > Final Cut Pro/Final Cut Express

2008-07-04

Contents

FxPlug SDK Overview

Developed by Apple's Professional Applications group, FxPlug is a compact yet powerful image processing plug-in architecture that lets you create new effects for Final Cut Pro, Final Cut Express, and Motion.

Leveraging technologies such as OpenGL, Quartz, Quartz Composer, and Core Image, you can develop unique plug-ins that include on-screen controls and custom UI elements—all running seamlessly in the host application. FxPlug supports both hardware-accelerated and CPU-based effects.

To write an FxPlug plug-in, you need to use the FxPlug SDK. The bulk of this SDK consists of Objective-C protocol definitions. You create a plug-in by writing code in Objective-C or Objective-C++ that conforms to these protocols. That is, you implement the methods declared by the protocols.

Note: After reading this document, an excellent way of learning the FxPlug framework is to study the examples installed in /Developer/Examples/FxPlug.

Versions of the FxPlug SDK

FxPlug SDK 1.0 was introduced together with Motion 2.0 in April, 2005. A few months later, version 1.0.2 was released with support for Universal Binary plug-ins but no API changes.

Version 1.1 of the FxPlug SDK was the first version that worked with Final Cut Pro 5.1.2 . Version 1.1 added a number of new features, including:

- Transitions.
- New FxBaseEffect parent protocol for filters, generators, and transitions.
- String parameters.
- A new `-properties` method that returns a dictionary describing plug-in attributes.
- 8-bit and floating-point YUV ('r408' and 'r4fl') bitmaps.
- Rowbytes support in bitmaps.
- SMPTE wipe equivalents for transitions.
- A new FxHostCapabilities class for determining the capabilities of the host application.

Version 1.2 of the FxPlug SDK was released with Final Cut Pro 6.0 and Motion 3.0. The features in this version of the SDK are detailed below in the section "Features of FxPlug SDK 1.2" (page 12).

Version 1.2.1 of the FxPlug SDK was released with Final Cut Pro 6.0.1 and Motion 3.0.1. The features of this version of the SDK are described below in the section "Features of FxPlug SDK 1.2.1" (page 17)

Version 1.2.2 of the FxPlug SDK was released with Final Cut Pro 6.0.2 and Motion 3.0.3. The features of this version of the SDK are described below in the section "Features of FxPlug SDK 1.2.2" (page 19)

Version 1.2.3 of the FxPlug SDK provides minor bug fixes and documentation corrections.

Backward Compatability

In general, to make your plug-in work with any version of a host application, have the plug-in first check for availability of a host API for an SDK feature, and then only use that feature if the API is available. If a particular host API is not available, your plug-in needs to fall back to an alternative behavior.

Plug-in Concepts

An FxPlug plug-in is a type of ProPlug, which in turn is a flavor of CFBundle. ProPlug is a generalized Apple architecture for writing plug-ins that can interact with a host application and extend its functionality.

When a plug-in needs to access host application functionality—such as requesting video stream information—it uses methods in a host API. A host API is an object, provided by the host application, that implements methods that can be called by a plug-in. It is analogous to a callback suite in other plug-in architectures, but is implemented as an Objective-C protocol. For more information about protocols, see the Protocols section in *The Objective-C 2.0 Programming Language*.

For example, the FxPlug SDK defines host protocols that include methods for requesting layer information, converting between canvas and object coordinate spaces, building a list of plug-in parameters, getting and setting parameter values, and evaluating input images at arbitrary times. Not all hosts are guaranteed to support all host-API protocols specified in FxPlug, so a plug-in must request the host API object before invoking its methods.

SDK Headers

The principal header for the FxPlug SDK is `FxPlugSDK.h`, which is located in `/Library/Frameworks/FxPlug.framework/Headers`. It imports all of the other FxPlug headers. The comments in the header files generate the reference documentation for the FxPlug SDK.

Types of Plug-ins

You can write three types of plug-ins with the FxPlug SDK: video filters, video generators, and video transitions. A filter operates on an input video image to produce an output video image. A generator does not require an input image. A transition combines two input video images to yield an output video image. In each case, the video images may be traditional RAM-based bitmaps or hardware-accelerated OpenGL buffers. A single plug-in can support either software rendering or hardware rendering, or both. Implementing a software path is always recommended to provide compatibility with older machines that may not have Quartz-Extreme-compatible video hardware.

Bundling and Installing a Plug-in

The easiest way to build a new FxPlug plug-in is to create a New Project in Xcode and use one of the FxPlug templates in the Standard Apple-Plug-ins category: FxPlug Filter, FxPlug Generator, or FxPlug Transition.

For examples of complete FxPlug plug-ins, see the directory `/Developer/Examples/FxPlug`.

Note: To avoid plug-in conflicts, you must edit the UUID entries in the `Info.plist` file. You can create unique UUIDs by issuing the `uuidgen` command in the Terminal application. (See the "Frequently Asked Questions" section below for details on using `uuidgen`.)

Two Mac OS X folders are reserved for FxPlug installation. When scanning for plug-ins, host applications recursively search the folder hierarchy from these locations:

```
/Library/Plug-Ins/FxPlug/
~/Library/Plug-Ins/FxPlug/
```

As you develop your plug-in and test it with the host application, you may find it convenient to perform the installation via a Copy Files build phase. Alternatively, you can create a symbolic link inside `/Library/Plug-Ins/FxPlug/`, pointing to the build-product directory specified in Xcode's Project Preferences. Depending on your settings, you can create the symbolic link in the Terminal application like this:

```
% ln -s ~/build/MyPlug.fxplug /Library/Plug-Ins/FxPlug/MyPlug.fxplug
```

Packaging

A completed FxPlug plug-in is packaged in a CFBundle. Each bundle contains one or more plug-ins. A bundle also contains an `Info.plist` property-list file that describes the plug-ins in the bundle. This description can be static or dynamic. With a static list, the plug-ins are recognized and loaded automatically; for a dynamic list, the bundle's principal class is asked to register the plug-ins. The distinction depends on the value of a Boolean tag `ProPlugDynamicRegistration` in the property list. Dynamic registration incurs a plug-in scanning performance hit. So you should ordinarily use static registration. You can find more information about dynamic registration in `<PluginManager/PROPlugInBundleRegistration.h>`.

Registration

Before a host application can display a list of available plug-ins, it recursively scans the plug-in folders for available CFBundles that conform to the FxFilter, FxGenerator, or FxTransition protocol. Plug-in properties such as name, group name, description, and so on are specified in the bundle's property list for statically registered plug-ins. The host application can display the list of grouped plug-ins in its UI, and allow the user to choose and apply particular plug-ins to particular tracks.

API Overview

Protocols

The primary protocols defined by the FxPlug SDK are FxFilter, FxGenerator, and FxTransition. Each plug-in conforms to one of these protocols. For more information about the methods in these protocols, see the reference documentation for the header files `FxFilter.h`, `FxGenerator.h`, and `FxTransition.h`. Note that these protocols inherit methods from a superprotocol FxBaseEffect, which is defined in `FxBaseEffect.h`.

Properties

The FxPlug SDK 1.1 introduced the plug-in properties dictionary. One of the methods defined by the new FxBaseEffect protocol was `-properties`. From this method, a plug-in returns an NSDictionary with key-value pairs that describe the capabilities of a plug-in.

Parameters

Once a plug-in is applied, the host application asks for its parameter list by calling the `-addParameters` method. This method is declared in the FxBaseEffect protocol, which is inherited by the FxGenerator, FxFilter, and FxTransition protocols. The host application then displays the parameters with the appropriate UI.

In the `-addParameters` method, a plug-in adds its parameters, one by one, using methods in the FxParameterCreation protocol. First, the plug-in calls the `-apiForProtocol:` method, defined by the PluginManager framework, to obtain the host's API object that implements the protocol. Then it calls the parameter creation methods to add the parameters. For example, this is an `-addParameters` method from a simple opacity filter:

```
-(  BOOL  ) addParameters
{
   id  <FxParameterCreation> paramsApi =
   [_apiManager apiForProtocol:@protocol(FxParameterCreationAPI)];

   if  ( paramsApi != NULL  )
   {
       [paramsApi addFloatSliderWithName:@"Opacity"
                             parmId:  OPACITY_ID
                       defaultValue:  1.0
                       parameterMin:  0.0
                       parameterMax:  3.0
```

```
                          sliderMin:  0.0
                          sliderMax:  1.0
                              delta:  0.01
                          parmFlags:  kFxParameterFlag_DEFAULT];
        return YES;
    }
    else
         return NO;
}
```

You can find more information about adding parameters in the reference documentation for the header files `FxParameterAPI.h`, `FxOptionalParameterAPI.h`, and `FxOptionalParameterAPI.h`.

Protocols in these headers include methods for adding these standard types of parameters:

- Floating-point slider
- Integer slider
- Angle slider
- Toggle button (checkbox)
- RGB color
- RGBA color
- Point
- Popup menu
- Image reference
- Group start
- Group end
- String

Note: String parameter support was added in FxPlug SDK 1.1. The methods for creating string parameters and retrieving and setting their values are defined in new protocols FxParameterCreationAPI_v2, FxParameterRetrievalAPI_v2, and FxParameterSettingAPI_v2. These new "_v2" protocols inherit all of the methods in their parent protocols, FxParameterCreationAPI, FxParameterRetrievalAPI, and FxParameterSettingAPI. If you are using string parameters, you should use the new _v2 protocols. Otherwise, you should use the older parent protocols, so that your plug-in will work in host applications that don't support the _v2 protocols.

There are also optional parameters that may be supported by some host applications but not by others:

- Histogram
- Gradient

In addition to the standard parameter types, you may create a custom parameter with an opaque data type, using the method:

```
-addCustomParameterWithName:parmId:defaultValue:parmFlags
```

Each custom parameter must be associated with a custom parameter view, which is defined by the methods in `FxCustomParameterUI.h`. More details about custom parameter UI can be found in the reference documentation.

Other protocols in these headers include accessor methods for getting and setting parameter values.

Rendering

In order to render an output frame for a filter plug-in, the host application calls the following sequence of plug-in methods:

```
-(BOOL)frameSetup:(FxRenderInfo)renderInfo
    inputInfo:(FxImageInfo)inputInfo
    hardware:(BOOL*)canRenderHardware
    software:(BOOL*)canRenderSoftware;

-(BOOL)renderOutput:(FxImage *)output
    withInput:(FxImage*)inputImage
    withInfo:(FxRenderInfo)renderInfo;

-(BOOL)frameCleanup;
```

Generator and transition plug-ins render with a similar sequence, but with different numbers of inputs passed to their `-frameSetup` and `-renderOutput methods`.

In its implementation of the method `-renderOutput:...withInfo:`, a plug-in requests parameter values and renders an output frame. The output object is an instance of one of two subclasses of the FxImage class: FxBitmap or FxTexture.

Note: In floating-point rendering, it is important to avoid clipping values to the 0.0-1.0 range. The superblack and superwhite pixel values outside this range are valid in digital video.

Images

In its implementation of the method `-frameSetup:`, a plug-in specifies whether it can render in software on the CPU, in hardware on the GPU, or in both. If the plug-in requests both kinds of rendering, the host application decides which to use. For example, Motion is more likely to choose a hardware path, while Final Cut Pro tends to prefer software rendering.

When rendering in software, input and output images are FxBitmap objects; in hardware, FxTexture objects. You can find more information about these image classes in the reference documentation for the header files `FxImage.h, FxBitmap.h`, and `FxTexture.h`.

Parameter Interaction

When a user changes a parameter control for an FxPlug plug-in, the host application calls the plug-in method `-parameterChanged:`. In response, the plug-in can change the state of other parameters. For example, if the value of a toggle button changes, the plug-in can hide or reveal other parameter controls, or change their values.

Note: In FxPlug 1.0, the `-parameterChanged:` method was included in the FxFilter protocol, but not in the FxGenerator protocol. In FxPlug 1.1, the `-parameterChanged:` method was moved to the FxBaseEffect protocol and is now inherited by all three types of FxPlug plug-ins. For applications such as Motion 2.0 through 2.1.x that do not support this new feature in FxPlug SDK 1.1, you can use custom parameter UI to create a generator plug-in that responds to parameter changes.

Custom Parameter UI

For a custom UI parameter, user interaction is handled differently. In this case,the plug-in conforms to theFxParameterViewHost protocol by implementing a `-createViewForParm:` method, to provide an NSView subclass to the host app. Like any other NSView subclass, that custom view draws itself in a window, and receives notification of user actionssuch as mouse, key, and tablet events.

In response to user events, a plug-in custom view can notify the host application that a parameter value has changed. A plug-in can create a custom view object programmatically, or may retrieve it from a NIB file created in Interface Builder and placed in the plug-in's Resources folder.

Note: In the plug-in's custom view methods, you must call the FxParameterAction host API method `-beginAction:` before your plug-in accesses any parameter values, and `-endAction:` after it has finished accessing them. This lets the host application set up and restore the internal state. See the reference documentation for the header file `FxCustomParameterUI.h` for more details about user interaction with custom parameters.

On-Screen Controls

In addition to standard and custom parameter UI, a plug-in can also implement custom on-screen controls that are composited directly onto the host application's canvas window using OpenGL. Examples of on-screen controls can be found in the SimplePaint example (`/Developer/Examples/FxPlug`) and in some of Motion's built-in filters such as Kaleidoscope and Basic 3D.

To create an on-screen control, you create a second ProPlug plug-in class that conforms to the FxOnScreenControl protocol. You package this class in the same bundle as its associated filter, generator, or transition plug-in.

Note: On-screen controls are supported by Motion 2.0 through Motion 2.1.x, but not by the new version of Final Cut Pro. To work around this limitation, you might check at runtime for the availability of the FxOnScreenControl API. If it is unavailable, you can use a custom parameter UI instead.

Host Capabilities

A new class in FxPlug 1.1 defines methods that your plug-in can use to identify key capabilities in the host application. The FxHostCapabilities class is specified in the new header file `FxHostCapabilities.h`. Using the methods in this class, you can tailor the behavior of your plug-in to the context in which it is operating. Example methods include `-upscalesFields` and -supportsHiddenParameters. For more details, see the reference documentation for the `FxHostCapabilities.h` header.

Note that this new class was not implemented in earlier versions of the FxPlug framework. If you reference the FxHostCapabilities symbol in your plug-in, the plug-in will only link if the version of the installed FxPlug framework is 1.1 or later. Your plug-in installer should check that FxPlug 1.1 or later is present. (If it's not, you can alert the user to run Software Update). Alternatively, you can retrieve the class by name using a string literal, as in this example:

```
Class class = NSClassFromString( @"FxHostCapabilities" );
id hostCaps = [[class alloc] initWithAPIManager:_apiManager];
if ( ![hostCaps supportsDisabledParameters] )
    ; // Do something
```

Features of FxPlug SDK 1.2

The focus of FxPlug SDK 1.2 is improved consistency between Motion and Final Cut Pro, and better support for timing information. In addition to the changes in the FxPlug framework itself, there are other FxPlug-related changes in Plug-in Manager 1.7, Final Cut Pro 6, and Motion 3.

The FxPlug 1.2 Framework

Timing information

A FxTimingAPI protocol defines the methods provided by the host that allow a plug-in to query the timing properties of its input image(s), image parameters, effect, timeline, and in/out points. This protocol is the most significant change in FxPlug SDK 1.2.

Field and Field Order Information

FxImage objects now respond to `-field` and `-fieldOrder` accessors. These accessors provide information about fields in interlaced images: the field identifier for an image, and the field order of an image. The incorrectly named field `FxRenderInfo.field` has been renamed `FxRenderInfo.fieldOrder`. But you should use the FxImage accessors instead.

Progress and Cancellation

A FxProgressAPI protocol defines methods for plug-ins that render slowly to update a progress bar and support user cancellation.

Image Retiming in Transitions

Final Cut Pro only: A FxTemporalTransitionImageAPI protocol allows a plug-in to retrieve its A or B input images at different times. In FxPlug 1.1, retiming was only enabled for images from filter inputs and image wells, not for transition input images.

Absolute versus Relative Times

A `[FxHostCapabilities timeBase]` method allows a plug-in to determine whether the host application measures times as frame offsets from the start of the timeline, or from the start of a clip, generator, filter, or transition.

Getting Angle Values

The `-getAngle:fromParm:atTime:` method is deprecated. Plug-ins should use `-getFloatValue:` instead.

Plug-in Manager 1.7

Free Access to Host APIs

The Plug-in Manager no longer requires a plug-in Info.plist file to declare which host APIs the plug-in might use. However, for a plug-in to load on a system with an older version of the Plug-in Manager, the plug-in should still list these APIs in the ProPlugProtocolList—as illustrated by the example projects and Xcode templates.

Final Cut Pro 6

Support for Hidden and Disabled Parameters

Final Cut Pro 6 respects `kFxParameterFlag_HIDDEN` and `kFxParameterFlag_DISABLED`. A plug-in can specify these flags when it creates parameters, and change them dynamically with the `-setParameterFlags:` selector of FxParameterSettingAPI.

Support for Non-Animatable Parameters

Final Cut 6 respects the `kFxParameterFlag_NOT_ANIMATABLE` flag if it is set when a parameter is created. A plug-in cannot change this flag dynamically once a parameter is created.

Support for Parameter Groups

Final Cut 6 provides some support for parameter groups. Parameter groups are implemented as labels separating the parameters, similar to After Effects plug-ins in Final Cut Pro. The groups cannot be collapsed or nested.

Support for FxParameterSettingAPI

Final Cut Pro 5.1.2 did not implement the FxParameterSettingAPI and a plug-in could not set the values of its parameters after creation. Final Cut 6 supports this API. Plug-ins can now change the values of their parameters dynamically. For example, an effect might want to implement a "presets" functionality by changing some parameter values when the user makes a choice from a popup menu.

Parameters Sampled at Arbitrary Times

A plug-in running under Final Cut Pro 5.1.2 could only sample parameters at the current render time. The only exception to this was image parameters. In Final Cut Pro 6, a plug-in can sample any type of parameter at any time. This allows an effect to examine the values of keyframed parameters at multiple times.

Plug-ins Only Instantiated Once at Startup

At startup, Final Cut Pro 5.1.2 instantiated each installed plug-in multiple times, sometimes sending the plug-in an `-initWithAPIManager:` message, sometimes sending the plug-in an `-init` message. At startup, Final Cut Pro 6 instantiates a plug-in once via `-initWithAPIManager:`, sends the instance the following messages:

- `-addParameters`
- `-variesOverTime`
- `-properties`

and releases the plug-in instance.

Plug-ins No Longer Instantiated Multiple Times

Final Cut Pro 5.1.2 created two instances of a plug-in when the plug-in was added to a timeline, using one instance for rendering and the other instance for parameter management. Final Cut Pro 6 creates one instance when a plug-in is added to a timeline and uses this single instance for both rendering and parameter management.

Initial Value of Point Controls Respected

Final Cut Pro 5.1.2 always set the initial value of point controls at the center of the image. Final Cut Pro 6 uses the value specified by the plug-in.

Software Rendering Preferred in More Cases

Final Cut Pro 5.1.2 asked a plug-in to render on the GPU unless the plug-in returned `canDoHardware = NO` from `-frameSetup`. Final Cut Pro 6 asks a plug-in to render in software if:

> The plug-in returns `canDoSoftware=YES` from `-frameSetup`
> AND
> The plug-in specifies `kFxPropertyKey_SupportsRowBytes = YES`

Otherwise, Final Cut Pro 6 asks the plug-in to render in hardware.

Consistent Parameter Change Notification

Final Cut Pro 5.1.2 sent plug-ins the `-parameterChanged:` message only if the parameter change caused a render. If the playhead was not on the item with the effect, Final Cut Pro 5.1.2 did not send the message. Final Cut Pro 6 sends a plug-in this message immediately when a parameter changes, whether the change causes a render or not.

Requesting a Nonexistent Parameter

In Final Cut Pro 5.1.2, a plug-in request for the value of a parameter that had not been added created an Objective-C exception and left Final Cut Pro in an inconsistent state. In Final Cut Pro 6, a request for a parameter that does not exist simply returns NO.

Output Pixel Aspect Ratio Reported Correctly

In Final Cut Pro 5.1.2, if an FxPlug effect was rendering in software and did not change the size of the output image, the output image would always have a pixel aspect ratio of 1.0. In Final Cut Pro 6, the pixel aspect ratio is reported correctly.

Remaining Issues in Final Cut Pro 6

No Support for Onscreen Controls

Final Cut Pro 6 does not support onscreen controls.

No Support for Resizing Output Images

Final Cut Pro 6 does not allow a filter to resize its output image. In addition, it no longer calls the `-getOutputWidth:height:` method.

Some APIs Available Only During Render

In Final Cut Pro 6, the FxTimingAPI and FxTemporalImageAPI only work correctly if called during one of the following selectors:

- `-getOutputWidth: height:`
- `-frameSetup`
- `-renderOutput`
- `-frameCleanup`

As well, sampling keyframed parameters at arbitrary times only works during these selectors.

Motion 3

Now Notifying Parameter Changes

Motion 2.1 did not call `-parameterChanged:` when the value of a compound parameter (e.g. point, color, histogram, gradient) changed. Motion 3 does.

Fixed Pixel Aspect of Textures from Image Wells

Motion 2.1 did not provide the correct pixelAspect value for FxTexture images retrieved from image well parameters. Motion 3 corrects this problem.

Float Images pixelFormat Correction

Motion 2.1 ordered pixel components RGBA in floating-point images, but the `-pixelFormat` method returned ARGB. Motion 3 corrects this problem. You can determine whether floating-point images are mislabeled by querying `[FxHostCapabilities formatsFloatRGBABitmapsAsARGB]`.

Temporal Image Retrieval for Image Wells

Motion 2.1 had problems retrieving images from image well parameters at arbitrary times. The resulting images were at incorrect times and upside-down. Motion 3 retrieves these images correctly.

Support for Non-Animated Parameters

Motion 3 adds support for the `kFxParameterFlag_NOT_ANIMATABLE` flag.

Scale Factor and Resizing Reconciled

Motion 3 corrects an issue in Motion 2.1 that produced incorrect results when resizing either via `-getOutputWidth:height:` or down-sampling for low quality render.

Features of FxPlug SDK 1.2.1

The FxPlug SDK 1.2.1 includes two new protocols, one for managing 3D camera and layer information in Motion, and another that allows plug-ins to manage backward compatibility issues. The host applications Final Cut Pro 6.0.1 and Motion 3.0.1 also contain fixes related to the FxPlug APIs.

The FxPlug 1.2.1 Framework

3D Support

Motion only: Plug-ins can use the new Fx3DAPI protocol to get the 3D transforms for the camera and for the plug-in's layer.

Versioning Support

Plug-ins that include a "version" key in their Info.plist files can use the new FxVersioningAPI protocol to determine what version of their plug-in was used when a project was created.

Angle Units

For angle parameters, `-getFloatValue` and `-setFloatValue` now use degrees in all host applications.

New Examples

In addition to SimpleMatte and SimplePaint, the FxPlug SDK 1.2 provided new example plug-in projects installed in /Developer/Examples/FxPlug:

- DirectionalBlurExample
- Options Dialog
- ScrollingRichText
- Slow SolidColor

Final Cut Pro 6.0.1 Changes

Correct Duration for Transition Effects

The method `[FxTimingAPI durationForEffect]` returns the correct duration of FxPlug transition effects.

Correct Start Time and Duration for Untrimmed Media

The methods `[FxTimingAPI startTimeOfImageParm]` and `[FxTimingAPI durationOfImageParm]` return the correct start time and duration for image parameters, even if the media in the image wells do not have in and out points set.

Group Parameters Can Be Hidden

The `kFxParameterFlag_HIDDEN` flag works correctly with group start and end markers.

Parameter List No Longer Scrolls Inappropriately

Hiding or showing a parameter no longer causes the parameter list to scroll to the top.

No Artifacts in Parameter List

Hiding a parameter no longer causes a spurious horizontal line to be drawn in the parameter list.

Correct Channel Order

Previously, the output images for RGB-only generators would be labeled as RGBA, but the results would be interpreted as ARGB in 8-bit. The images are now correctly labeled as ARGB in 8-bit and RGBA in float.

Motion 3.0.1 Changes

No Gamma Shift

Retrieving bitmap images using the temporal image API no longer causes a gamma shift in the retrieved image.

Correct Rendering

Retrieving bitmap images from a group no longer causes the group to render upside down.

Correct Memory Management

Motion now properly releases custom NSViews when a filter is deleted.

Proper Bit Setting

Retrieving bitmap images from 16-bit per channel footage now returns proper 32-bit per channel footage.

Avoid Unnecessary Rendering

Repeated renders are no longer queued up when a custom control calls `-startAction` and `-endAction` during its `-drawRect:` method. This keeps the control and the current frame from constantly rerendering.

Fix Crash Bug

Motion no longer crashes under certain circumstances when calling `-currentTime:` in the FxCustomParameterActionAPI protocol.

Parameters Updated Correctly

Parameters are now properly updated in the inspector when their enabled-disabled or hidden-shown state is changed.

Features of FxPlug SDK 1.2.2

The FxPlug 1.2.2 Framework

FxImageColorInfo Enumeration

The new `FxImageColorInfo` enumerated type identifies some basic color properties of an FxImage. For YUV images, this enumeration tells you whether you should use the Rec. 601 or Rec. 709 color matrix to convert an image to RGB. For RGB images, it describes the gamma level of the image.

Xcode Template Location

FxPlug SDK 1.2.2 installs its Xcode templates in the location expected by Xcode 2.5 and later. Symbolic links are installed into the legacy location, so you can still open the templates using an earlier version of Xcode.

Final Cut Pro 6.0.2 Changes

Parameter Value Retrieval

Plug-ins can now evaluate their parameter values at any time during `-parameterChanged`, or from within custom UI code. Previously, parameters could be evaluated at any time during `-renderOutput`, but otherwise only at the current time.

Collapsible Groups

Parameter groups now contain disclosure triangles that allow a group to be collapsed and expanded. The `kFxParameterFlag_COLLAPSED` flag is also respected; plug-ins can use this flag to create groups that are initially collapsed, or to programmatically collapse or expand groups in response to other parameter changes.

Start Point, End Point and Reverse Transitions

The start point, end point, and reverse controls in the Transition Viewer are now respected when rendering FxPlug transitions. These three controls affect the "time fraction" passed to the transition's render method; the "reverse" control will also swap the A and B images. You should not have to change your plug-in to take advantage of these controls.

Undo and Copy/Paste Crashes

Fixed a bug that could result in crashes when undoing the addition of a filter, or when copying and pasting filters.

Nested Groups

Hiding and disabling groups now work correctly with nested groups. However, there is still no visual indication that the groups are nested.

YUV Images

Plug-ins can use the new `FxImageColorInfo` API to determine if a YUV image is Rec. 601 or Rec. 709. This allows for improved YUV support, or accurate conversion to high-precision RGB within the plug-in.

FxTemporalTransitionImage

Previously, the `FxTemporalTransitionImage` API interpreted the times passed as relative to the requested source clip. Now the times are interpreted as relative to the transition item. This is consistent with all other uses of time in Final Cut Pro.

Multiple Monitors

Fixed a bug where custom UI could appear in the wrong location if the filter viewer was dragged to a second monitor.

Legacy Non-Real-Time Display Path

Fixed a bug where input images in the legacy non-real-time display path were not tagged with the correct aspect ratio.

Transition Input images now have the correct field order

Fixed a bug where input images for transitions were not labeled with the correct field order.

Hidden Groups

Final Cut no longer ignores the `kFxParameterFlag_HIDDEN` flag when creating groups. However, groups must be disabled after the parameters they contain have been added.

Motion 3.0.2 Changes

Custom Control Saving

Previously, when a user changed a document with a custom control, Motion did not flag the document as changed and could close it without saving the changes. Motion now properly flags the document as changed. It also fixes a crash that could occur during an undo.

Image Well Gamma Shift

When you retrieve a bitmap from an image well parameter, the image now has the proper gamma setting that matches the gamma setting it would have if you had dropped it directly into the timeline.

API Object Refactoring

The various API Objects have been refactored to make them accessible at more times and to reduce the number of thread related problems. Plug-in developers should test to be sure that the APIs all still work as expected.

Interpolation for Upsampling

Motion now uses interpolation rather than line doubling to upsample fields to frame size.

FAQs

How can I debug my plug-in?

To test your plug-in code in a target application, you can choose Project > New Custom Executable in Xcode and select Final Cut or Motion. Then choose Run > Debug to launch the selected host in the Xcode debugger.

How can I retrieve resource files from my plug-in bundle?

You can put the file in the Resources folder for the plug-in. Then you can access the bundle from a method of one of the classes in your plug-in, like this:

```
  NSBundle *bundle = [NSBundle bundleForClass:[self  class]];
```

(The FxPlug Xcode templates does this to get localized strings.) To get data from a file called "filename.extension" in your Resources folder, you can use this snippet:

```
NSBundle *bundle = [NSBundle bundleForClass:[  self  class]];
NSString *path = [bundle pathForResource:  @"filename"
ofType:  @"extension"  ];

 NSString *dataString = [NSString stringWithContentsOfFile:path];
```

Copying and pasting UUIDs from Terminal is tedious. How can I simplify this step?

You can create an Xcode script to speed this up. To do this in Xcode 3.0, follow these steps:

1. Select `Edit User Scripts` from the `User Scripts` menu (identified by the script icon in the Xcode menu bar).
2. Select `Code` from the list of script types.
3. Click the `Add` button and select `Shell Script`.
4. Double click the name `Shell Script` and revise it to `GenUUID` or some other appropriate name.
5. Add the line ``echo -n `uuidgen` `` to the script .
6. Change the `Output` option to `Replace Selection`.
7. Close the `Edit User Scripts` window.

You can now generate a UUID in Xcode by choosing your new script from the `Code` submenu of the `User Scripts` menu.

How can I determine if an input frame is interlaced?

If a frame is interlaced, you'll know in your `-renderOutput:` method, because the `FxRenderInfo` structure that's passed in will have the "field" member set to either `kFxField_UPPER` or `kFxField_LOWER`. Note that it doesn't tell you the field you have, but rather what the field order is. If a frame is not interlaced, then the "field" member will have the value `kFxField_NONE`.

How can I access a single field from an interlaced frame?

First, get the FxTemporalImageAPI and call one of these two methods:

```
-getInputBitmap:withInfo:atTime:
```

or

```
-getInputTexture:withInfo:atTime:
```

For the first field of frame *t*, use time *t*, and for the second field, use time *t + 0.5*.

How can I determine what application my plug-in is running in?

The following code snippet returns a string, `appIdentity`, which specifies the host application:

```
CFBundleRef appBundle = CFBundleGetMainBundle();
CFStringRef appIdentity = NULL;
if ( appBundle != NULL )
    appIdentity = CFBundleGetIdentifier( appBundle );
```

Motion's bundle identifier is `com.apple.motion`; Final Cut Pro's is `com.apple.finalcutpro`.

How can I specify another clip as input?

In FxPlug parlance, a reference to another clip or external media file is called an image reference. So, using the FxParameterCreationAPI protocol, call the following method to create the parameter:

```
-addImageReferenceWithName:parmId:parmFlags:
```

To get the parameter's value, use either:

```
-getBitmap:layerOffsetX:layerOffsetY:requestInfo:fromParm:atTime:
```

or

```
-getTexture:layerOffsetX:layerOffsetY:requestInfo:fromParm:atTime:
```

How can I get the current time?

In your `-parameterChanged:` method, and in response to an event in a custom parameter view or on-screen control, you'll need to determine the current time. You can do this by calling the `-currentTime` method in the FxCustomParameterAPI protocol.

Can I define multiple plug-in bundles that populate the same group?

Yes, but the Plug-in Manager places a restriction on how you do this. If you create multiple FxPlug effects in different bundles, and assign them to the same groups, the different bundles must use different group UUIDs, but the same group name. If you use the same group UUID in two bundles, the plug-ins in only one of the two bundles are loaded.

Document Revision History

This table describes the changes to *FxPlug SDK Overview*.

Date	Notes
2008-07-04	Minor corrections for FxPlug SDK 1.2.3.
2007-11-16	Update for FxPlug SDK 1.2.2
2007-06-28	Update for FxPlug SDK 1.2.1
2007-05-21	Update for FxPlug SDK 1.2.
2006-09-26	Update for FxPlug SDK 1.1, to cover support for Final Cut Pro.
2005-06-04	New document that describes how to write plug-ins for Apple professional applications. FxPlug is currently supported by Motion 2.0.