Adobe Premiere Pro CS5

Software Development Kit

Adobe Premiere Pro CS5 Software Development Kit, Pre-Release 1

Version History

13 February 1995	Matt Foster, Nick Schlott	Version 4.0 - first Windows release
9 February 1996	Brian Andrews	Version 4.2
20 April 1998	Brian Andrews	Version 5
10 December 2000	Bruce Bullis & Eric Sanders	Version 6 release 1
10 May 2001	Bruce Bullis	Version 6 release 2
19 July 2002	Zac Lam & Bruce Bullis	Version 6.5
21 August 2003	Zac Lam	Version 1.0 (Premiere Pro)
25 May 2004	Zac Lam	Version 1.5
17 January 2006	Zac Lam	Version 2.0 release 1
13 July 2006	Zac Lam	Version 2.0 release 2
5 October 2007	Zac Lam	Version CS3
21 September 2009	Zac Lam	Version CS4
28 April 2010	Zac Lam	Version CS5

Table of Contents

1. Introduction

2. Resources

3. Universals

4. Hardware Integration

5. Importers

6. Recorders

7. Exporters

8. Players

9. Transitions

10. Video Filters

11. Device Controllers

1 Introduction

Welcome to the Adobe® Premiere® Pro CS5 Software Development Kit! This is a living document, and is constantly being updated and edited. The latest release of the SDK is available at: http://www.adobe.com/devnet/premiere/

If you have questions about the APIs described in this document, or about integration with Premiere Pro, your question may already be answered on the Premiere Pro SDK forum at: http://forums.adobe.com/community/premiere/premierepro_current/sdk.

SDK Audience

The Premiere Pro Software Development Kit enables developers to create plug-ins for Premiere Pro, After Effects, Encore, Soundbooth, Media Encoder, and Premiere Elements.

The required development environment for the Premiere Pro SDK for Windows is Microsoft Visual Studio .NET 2008 SP1 on Windows Vista 64-bit and Windows 7 64-bit. On Mac OS, the required environment is XCode 3.1 on Mac OS 10.5, or XCode 3.2 on Mac OS 10.6 . The SDK includes sample projects for these development environments. On Windows, projects can often be updated to more current versions of Microsoft Visual Studio .NET by simply opening the project and approving the automatic conversion. The sample code is written in C++. Other compilers and programming languages are not supported. We cannot assist with platform API programming issues not central to Premiere Pro plug-in programming.

Having a solid understanding of digital video concepts is vital to developing plug-ins. This documentation assumes you understand basic video topics such as resolution, frame rates, field interlacing, pixel aspect ratio, bit depth, timecode, compression, color spaces, etc. You must also understand how your plug-in will fit into a user's workflow in Premiere Pro. If you aren't yet familiar with Premiere Pro or video editing concepts, we recommend the Adobe Premiere Pro Classroom in a Book.

What Premiere Plug-ins Do

Premiere APIs provide access to many points of the video editing pipeline. Recording from an external device, device control, file import, video effects and transitions, playback, and output can all be performed by plug-ins. Hardware acceleration is enabled by writing plug-ins to manage various aspects of media handling.

If this is your first time developing a Premiere plug-in, you can skip the What's New section, and go directly to Where Do I Start?

What's New?

What's New in CS5?

Premiere Pro is now a 64-bit application! This is the single most important change that affects plug-ins. As a result of the 64-bit port, we have had to change code that assumed a `long` was 32-bits. CS5 does not support 32-bit plug-ins developed with the CS4 or earlier SDKs. Plug-ins for CS5 must be built using the CS5 SDK. The sample projects have been ported to 64-bit on Windows and Mac OS.

Importers now have access to the resolution, pixel aspect ratio, timebase, and audio sample rate of the source clip from a setup dialog. Custom importers can use a new call to update a clip after it has modified by the user in the setup dialog. Please refer to the Importers chapter for more info on what's new.

Recorders can now provide audio metering during preview and capture. Read more about what's new in the Recorders chapter.

Exporters and ***players*** can automatically take advantage of GPU acceleration, if available on the end-user's system. Each project now has a setting for the renderer that the user can choose in the project settings dialog. When renders occur through the Sequence Render Suite or the Playmod Render Suite, they now go through the renderer chosen for the current project. This allows third-party exporters and players to use the built-in GPU acceleration available in the new Mercury Playback Engine.

Exporters and players can now handle any pixel format, with the new Image Processing Suite. Exporters and players that parse segments and perform their own rendering can now call the host for subtree rendering. See the Video Segment Render Suite for details.

If you provide an installer for an exporter, note that custom presets created in Premiere Pro are now visible in AME and vice-versa.

Players can respond to the new field display settings, and play/pause resolution settings, in the Monitor panels. Please refer to the Players chapter for more details on what's new.

Video filters, in the Effects panel, can now appear with badges to advertise if they support YUV, 32-bit, and accelerated rendering. See more about what's new in the Video Filters chapter.

QuickTime VOut component support has been removed in CS5, due to current 64-bit limitations in QuickTime.

Quick Tips For 64-bit Windows Porting

1) Open your Visual Studio solution file in Visual Studio 2008. Perform the automatic conversion to upgrade the solution from a previous version.
2) Add a new solution platform of type "x64".
3) Now you'll need to port any code that assumed 32-bit compilation: Use the new cross-platform types in PrSDKTypes.h, update suite usage for suites that are no longer supported, and so on. Voila! The plug-in project is ready to compile a 64-bit binary!

Quick Tips For 64-bit Mac Porting

1) Open your XCode project file. In the project's info panel, in the General tab, set "Cross-Develop Using Target SDK:" to "Mac OS X 10.5".
2) In the Build tab, set the Architectures setting to build a 64-bit binary. Note that if you have already set Target-specific settings in the XCode project, this Architecture setting will not take effect, and you will need to open the target's info panel and set it there. You can confirm that the plug-in is being built for the correct architecture from the Terminal, by running "otool -f <filename>", where the filename is the binary inside the Contents/MacOS/ folder of the plug-in bundle.
3) For anything that includes Cocoa, you will need to set "Compile Sources As" to "Objective-C++".
4) Now you'll need to port any code that assumed 32-bit compilation: Use the new cross-platform types in PrSDKTypes.h, update suite usage for suites that are no longer supported, and so on. Voila! The plug-in project is ready to compile a 64-bit binary!

Encore CS5

Encore will remain a 32-bit application for CS5. So if you are developing plug-ins for Encore, use the CS5 headers to create 32-bit plug-ins. We have left the 32-bit configurations in the sample projects to facilitate this.

3rd-party exporters can now be used to transcode assets to MPEG-2 or Blu-ray compliant files. Please refer to the Guidelines for Exporters in Encore for instructions on how to set up your exporter so that Encore can use it for transcoding.

Mac 64-Bit and Cocoa

It is invalid to unload any bundle that uses Cocoa because of restrictions in the Objective-C runtime which do not support unregistering classes. If a plugin uses Cocoa, it must call CFRetain on its own bundle, otherwise it will cause a crash when the application is closing and tries to unload the plug-ins.

What's New in CS4?

New Exporter API

The new exporter API replaces the old compiler API as the way to export video, audio, and still images, and to generate preview files. In CS4, all forms of export from Premiere Pro use the Adobe Media Encoder UI for a unified export interface. If you are updating a legacy compiler plug-in to become a new exporter plug-in, most of the export code will remain the same, but the code that defines and makes use of user-controllable parameters will need to be updated. This lets your plug-in take a wide selection of customizable UI components that match the look of the built-in exporters. Your exporter can also be used within the standalone Adobe Media Encoder application. The old compiler API is no longer supported. Also, when an exporter is loaded in the standalone Adobe Media Encoder application, it does not have access to the full set of RT segments, so accelerated export must be implemented using both the exporter API and the new renderer API.

New Video Segments

The new video segments replace the old segments as the way to get information about sequences in the timeline. Rather than the host sending the video segments to the player, the player must request specific details using the Video Segment Suite. The new segments provide a hash value that the player can use to quickly determine whether or not a segment has changed. This hash value can be maintained even if a segment is shifted in time. The new segments also allow a player to accelerate segments with time remapping. Since the video segments have changed, `plugGetVideoSegmentFunc` and `plugDisposeSegmentFunc` have been replaced by the Video Segment Suite.

New Renderer API and Custom Pixel Formats

The new renderer API provides a way to take over and accelerate rendering of segments. Just as a player can choose which segments to accelerate, so a renderer can choose which segments to accelerate. Renderers may accelerate any segment, in any sequence, in any project.

Renderers also provide a way to add completely custom pixel formats to the render pipeline. Supporting a custom pixel format in an importer, a renderer, and an exporter is the new way to implement smart rendering, by passing custom compressed data from input to output.

Sequence Preview Formats

Sequence preview file formats are now defined by Sequence encoder preset files. Without any presets installed, you will not be able to create a new sequence using your custom editing mode.

Sequence-specific Settings

Sequences in the same project can now have different settings. So there is no longer a single editing mode per project. This means that the Editing Mode suite and most selectors for the `getSettings` callback in the utilFuncs are no longer supported. So if you are developing a recorder or device controller plug-in, you should provide reasonable defaults when the plug-in is initialized.

Also, project presets have been replaced by sequence presets. The difference from project presets are that sequence presets do not contain information to initialize capture settings.

Separate Importer Process

For CS4 only, importers are loaded and called from a separate process. As a result of being in a separate process, (1) all importers must do their own file handling, (2) `privateData` is no longer accessible from *imGetPrefs8*, and (3) the compressed frame selectors such as *imGetCompressedFrame* are no longer supported (this may now be achieved using custom pixel formats and a renderer plug-in).

To debug importers, attach to the ImporterProcessServer process. There is also a new Importer Process Plugin Loading.log.

Separate Processes During Export

When choosing export settings, the settings UI is displayed by Premiere Pro. When the user confirms the settings, the clip or sequence is passed to Media Encoder. From Media Encoder, frames from the clip or sequence can be retrieved and rendered without further participation from Premiere Pro. For a clip export, Media Encoder uses any installed importers to get source frames. For sequence export, Media Encoder uses a process called PProHeadless, to import and render frames to be exported.

Since there are so many processes involved during export, it is important that plug-ins be accessible to all processes, by being installed in the common plug-ins folder. PProHeadless Plugin

Loading.log provides information on the PProHeadless process. PProHeadless is also used when the user creates a dynamic link to a .prproj that is not opened in Premiere Pro.

XMP metadata

There are built-in XMP metadata handlers for known filetypes. These handlers write and read metadata to and from the file, without going through the importer. *imSetTimeInfo8* is no longer called, since this is set by the XMP handler for that filetype.

More Pixel Format Flexibility

Effects, transitions, and exporters no longer need to support 8-bit RGB at a minimum. So, for example, an effect can be written to process floating point YUV only. If necessary, Premiere will make an intermediate conversion so that the effect will receive the pixel format it supports.

New RT status

Players can now mark a segment yellow, so that it is not rendered when previewing the work area, but is rendered before export to tape.

Plug-in Blacklisting

Have a plug-in that works fine in one CS application, but has problems in another CS application? Now, specific plug-ins can be blocked from being loaded by MediaCore in specific applications, using blacklists. Note that this does not work for After Effects plug-ins loaded by AE, although it does work for AE plug-ins loaded in Premiere Pro. In the plug-ins folder, look for the appropriate blacklist file, and append the the filename of the plug-in to the file (e.g. BadPlugin, not BadPlugin.prm). If the file doesn't exist, create it first. "Blacklist.txt" contains names of plug-ins black-listed from all apps. Plug-ins can be blocked from loading in specific apps by including them in "Blacklist Adobe Premiere Pro CS4.txt", or "Blacklist Adobe After Effects CS4.txt", etc.

New Plug-in Support in Encore

Encore now supports third-party importers, players, and QuickTime VOut components.

What's New in CS3?

Mac OS Support

The only difference between the Mac and Windows SDKs are the development environment-specific project files for the sample projects. All plug-in types are supported on Mac OS and Windows. QuickTime VDig and VOut components are supported on Mac OS for video capture and video playback output. Core Audio is also supported on MacOS for audio playback output.

Some types have changed to enable smoother cross-platform development.

- All occurences of `wchar_t` have been changed to a Premiere-specific type, `prUTF16Char`, which is defined in `PrSDKTypes.h` as a `wchar_t` on Windows, but an unsigned short on MacOS. This is because `wchar_t` is 16-bit on Windows, but 32-bit on MacOS, and not well supported by MacOS APIs. By contrast, unsigned short is interchangable with `UniChar`, a basic MacOS type. `PrSDKTypes.h` also provides utility functions `prUTF16CharCopy` to replace `wcscpy`, `prUTF16CharCompare` to replace `wcscmp`, and `prUTF16CharLength` to replace `wcslen`.
- All occurences of `__int64` have been changed to a Premiere-specific type, `prInt64`, which is defined in `PrSDKTypes.h` as a `__int64` on Windows, but a `signed long long` on MacOS.
- The new utility function `prSetRect` sets the dimensions of a `prRect` struct. This should be used because MacOS `Rect` members have a different ordering than Windows `RECT` members.
- Note: The `prRgn` parameter of the `StretchBits` function in the Bottlenecks is only used on Windows.

Plug-in type specific changes

Importers, recorders, compilers, players, video filters, and device controllers have all had API changes that are documented within their specific chapter.

Creating XCode Projects From Existing Windows Code

1) Create a new Carbon Bundle project.
2) Replace main.c with your source files. Add the files relative to the project.
3) Add the headers to the External Frameworks and Libraries folder.
4.0) If the plug-in requires a PiPL (.r file), add a New Build ResourceManager Resources Build Phase
4.1) Add the .r file to the Build ResourceManager Resources Build Phase
4.2) Make sure the .r file includes the following:

```
#ifndef PRWIN_ENV
```

```
#define MAC_ENV
#include "PrSDKPiPLVer.h"
#include "PrSDKPiPL.r"
#endif
```

Premiere Plug-in Support in Other Production Premium Applications

Premiere importers are now supported in After Effects and SoundBooth. Premiere exporters are also supported in SoundBooth. See also a comprehensive chart of plug-in support across Production Premium applications.

The App Info Suite can be used by all Premiere and After Effects plug-in types to determine what host application they are running in.

Miscellaneous

The `getMarkerData` callback can now return the timebase of a marker.

Other plug-in specific changes are in the appropriate chapter for the plug-in type.

Legacy API

Legacy API features, such as selectors and callbacks that are superceded by new ones, are deprecated, but are supported, unless indicated.

Where Do I Start?

Read about the sample projects. Decide which one is closest to the functionality you want to provide. Build the plug-in into the shared plug-ins folder. Launch Premiere Pro with the debugger attached, and set breakpoints at the plug-in's entry point to see all communication between Premiere Pro and the plug-in. The documentation is intended as a reference with detailed explanation where appropriate, but studying the interaction between Premiere Pro and plug-ins is the best way to understand it.

Write plug-ins by modifying sample plug-in source code. This will greatly simplify your efforts, and make it easier for us to help you. Feel free to explore and experiment with the API on your own once you're familiar with it, but please, resist the temptation to start from scratch; you'll only force yourself to repeat other developers' mistakes, including our own.

If you run into behavior that seems wrong, see if you can reproduce the behavior using one of the unmodified sample projects. This can save you a lot of time, if you can determine whether the bug behavior was introduced by your modifications, or was already there to begin with.

Document Overview

This introduction information is common to all the plug-in types. All developers should read this chapter, as well as chapters two and three.

Chapter 2 describes the Premiere Pro-specific resources used by plug-ins, including the Plug-in Property List (PiPL).

Chapter 3, Universals, documents the data types and structures used throughout the APIs, and suites and functions available across different plug-in types.

Chapter 4 introduces Media Abstraction, used by hardware integrators and software developers to integrate with Premiere and accelerate specific workflows.

The remainder of the document describes specific plug-in types.

Documentation Conventions

Functions, structure names and general C/C++ code are in `Courier`;
`MyStruct.member` and `MyFunction()`
Underlined text in light blue is hyperlinked.
Premiere selectors are italicized; *imGetPrefs*.

Premiere Pro Plug-in Types

All Premiere Pro plug-ins are called in response to a user selection. There are currently no plug-in types to programatically control Premiere Pro.

Recorders	Records from a (usually hardware) source to disk. If necessary, provides a plug-in-defined settings dialog. Displays the video overlay in the preview area of the Capture panel. Any audio preview should be played directly be the recorder. The captured file is passed to Premiere after capture by its file path. The recorder can optionally provide the timecode of the captured file to Premiere Pro.
Importers	Import video and audio media into Premiere. Synthetic importers, a subset, dynamically synthesize media without creating an actual file on disk. Custom importers, dynamically synthesize media to disk.

Players	Provide all video output to the Monitor and optionally any external device during playback and editing. The player receives all information about the timeline, and can optionally accelerate video playback by providing an accelerated render path for of any segment of the timeline. The player can provide its own audio output, but using an ASIO driver is recommended. A player that provides its own audio output cannot support audio input for voiceover recording in the audio mixer, whereas an ASIO driver can.
Exporters	Allows the user to output media to disk. Can additionally be associated with a player and bound with an Editing Mode XML file to form an Editing Mode.
Video Filters	Process a series of video frames with parameters that can be animated over time. We strongly recommend using the After Effects SDK to develop effects plug-ins. Most of the effects included in Premiere Pro are After Effects plug-ins.
Video Transitions	Process two video sources into a single destination over time.
Device Controllers	Control an external device (video tape recorder, camera, etc.) during Capture and Export To Tape.
Other supported plug-in standards	
Adobe After Effects API	Premiere Pro supports a portion of the AE API. The After Effects SDK is not included in the Premiere Pro SDK. The last chapter in the After Effects SDK Guide.pdf, included in the After Effects SDK, contains information on known differences with how Premiere Pro supports the AE API.
VST	Premiere supports version 2.3 of the VST specification for audio effects.
ASIO	An ASIO driver is often provided in addition to a player, to provide audio output during editing, playback, and Export To Tape. An ASIO driver is required to support audio input for voiceover recording in the audio mixer. On Mac OS, a Core Audio component may be provided rather than an ASIO driver.
Core Audio	Mac OS only. May be provided instead of an ASIO driver.
QuickTime VDig	Mac OS only. Premiere Pro's built-in QuickTime recorder is a VDig host to capture (or digitize) from sources that support this API. VDig components will appear as an option in the QuickTime recorder's Video settings, in the Source tab.
QuickTime VOut	Mac OS only. Premiere Pro's built-in player is a VOut host to play video out to destinations that support this API. VOut components will appear as an option in the built-in player's Playback Settings, in the External Device drop-down menus.

Plug-in Support Across Production Premium Applications

This chart shows which third-party plug-ins are supported by the various Production Premium applications.

	Premiere Pro	After Effects	Encore	SoundBooth	Media Encoder
After Effects AEGPs		X			
After Effects effects	X	X			
ASIO	X	X	X	X	
Premiere device controllers	X				
Premiere exporters	X		X	X	X
Premiere importers	X	X	X	X	X
Premiere players	X		X		
Premiere recorders	X				
Premiere transitions	X				
Premiere video filters	X				
QuickTime codecs	X	X	X	X	X
QuickTime Export Components		X			
QuickTime VDig	(Mac OS)				
VfW codecs	X	X	X	X	X
VST audio effects	X				

Encore can use third-party exporters to transcode assets to MPEG-2 or Blu-ray compliant files. Please refer to the Guidelines for Exporters in Encore for instructions on how to set up your exporter so that Encore can use it for transcoding.

Premiere Elements Plug-in Support

Premiere Elements uses the same core libraries for plug-in support that Premiere Pro does. Premiere Elements 4 and 7 both use libraries equivalent to Premiere Pro CS3. Premiere Elements 8 uses libraries equivalent to Premiere Pro CS4. In many cases, a plug-in written for Premiere Pro will just work in Premiere Elements, but it's always important to test the plug-in fully in each application before advertising compatibility. Check out the Guidelines for Exporters in Premiere Elements for instructions on how to set up your exporter to be used in Premiere Elements.

What Is a Premiere Plug-in, Exactly?

Premiere plug-ins contain a single entry point of a type specific to each API. Plug-ins are DLLs on Windows, and Carbon or Cocoa Bundles on Mac OS. Plug-ins in the \Plug-ins\[language] folder,

and any of its subfolders, will be loaded at launch. Plug-ins can have private resources. Only one plug-in per file is parsed, unlike After Effects and Photoshop plug-ins, which can contain multiple entry points.

Sample Projects

Descriptions

Name	Description
SDK File Importer	This importer supports .sdk media files. To use the importer, choose File > Import, and select an .sdk file. Such files may be created using the SDK Exporter. It supports uncompressed 8-bit RGB with or without alpha, and packed 10-bit YUV (v410). It supports mono, stereo, and 5.1 audio at arbitrary sample rates and 32-bit float. It supports trimming using the Project Manager, Properties and Data Rate Analysis, Unicode filenames, the avoidAudioConform flag, and can read video frames asynchronously. It also features a test harness for multistream audio, which can be turned on by uncommenting the MULTISTREAM_AUDIO_TESTING define in the header.
Synth Import	This synthetic importer generates 8-bit YUV and RGB, video only. To use it, choose File > New > SDK Synthetic Importer. When the clip is created, it demonstrates a sample settings dialog, which can be displayed again by double-clicking the clip in the Project Panel or Timeline Panel. Every time the settings dialog is displayed, it creates new footage in memory. It creates ten seconds of footage at 24 fps. The video consists of horizontal lines of random colors. No file is created on disk - for an example of that, see the Custom Importer.
SDK Custom Import	This custom importer creates a clip similar to the Synth Import sample, but generates it to disk, rather than memory. To use it, choose File > New > SDK Custom Importer. Or, import an existing .sdkc clip from the File > Import dialog. On Windows, newly generated files with .sdkc file extensions are created in C:\Windows\Temp\. On Mac OS, they are created on the Desktop. After the sample settings dialog, it optionally displays a background frame from the timeline (useful for titlers). The generated footage is between 2 and 30 frames at 24 fps, with a random resolution between 32 and 720 pixels wide and between 32 and 480 high, at DV NTSC pixel aspect ratio.

Record	This recorder pretends to capture .sdk files. To select it, choose Project > Project Settings > Capture > Capture Format: SDK Record. To simulate a capture, there must be a valid .sdk file at C:\premiere.sdk, and the SDK File Importer must also be installed. When the record button is pressed, a capture is simulated, and when capture is finished, the file at C:\premiere.sdk is copied to the file specified in the Save Captured File dialog, and automatically imported into the project. It demonstrates a simple implementation of two capture options buttons, audio capture settings directly in the Project Settings > Capture panel, Unicode filenames, and changing the capture format mid-stream.
SDK Exporter	This exporter writes .sdk files. To use it, choose File > Export > Media, and in the Export Settings choose File Type: SDK File. It supports uncompressed 8-bit RGB with or without alpha, and packed 10-bit YUV (v410). It supports mono, stereo, and 5.1 audio at arbitrary sample rates and 32-bit float. It demonstrates custom parameters, including a custom settings button. It also writes marker data to an .html file with the same filename. To write files with v410 compression using 8-bit RGB sources, this sample uses routines to convert the 8-bit RGB data to 32-bit RGB, then to 32-bit YUV, and finally to v410. These same routines may be adapted for transitions, filters, and other plug-in types. This exporter can also be used with RTPlayback and the SDK Editing Mode.xml (in Examples\Editing Modes) to create a new editing mode. To select the editing mode, choose File > New > Sequence > General > Editing Mode: PlayerSDK. Render files created in this editing mode will be created using this exporter.

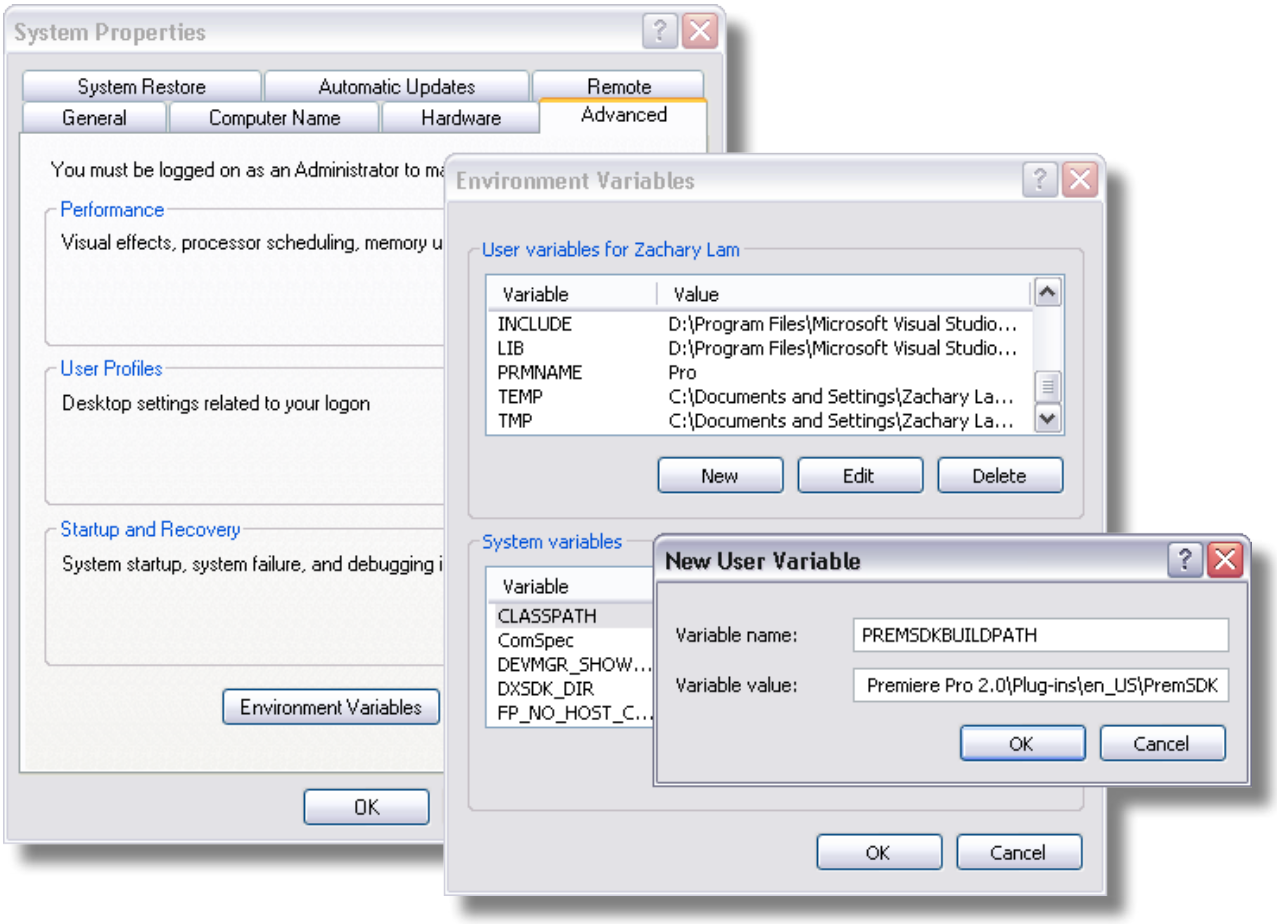
RTPlayback	This player is for use with the SDK Exporter and the SDK Editing Mode.xml (in Examples\Editing Modes) to create a new editing mode. To select the editing mode, choose File > New > Sequence > General > Editing Mode: PlayerSDK. It demonstrates cutlist management using a linked list structure, video playback with audio synchronization, scope rendering, segment parsing, and RT range reporting. It supports all the Monitor transport controls, the safe-area display, monitor window zoom levels, render quality settings, and the Multicam Monitor. On Windows, it overlays a description of the current segment on the Monitor, which is a great way to learn about segments. On Mac OS, the same information is calculated, but this player has not yet implemented that overlay. When the Playback Settings button is pressed, it displays a message box on Windows, and an alert on Mac OS. When the SDK Device Control is the current device control and the Timeline panel is active, File > Export > Export To Tape is active. Selecting this will first render any non-RT segments using the SDK Exporter, next the Device Controller will show its Export To Tape dialog, then this player will play the work area, and then the Device Controller will show its Export To Tape dialog again.
Transition	This transition implements a simple cross dissolve. It supports 8-bit and 32-bit YUV and RGB. It is found in the SDK folder of the Video Transitions in the Effects Control panel. It uses the `StretchBits` utility callback for 8-bit processing. It demonstrates usage of `instanceData`, and `FXCallbackProcPtr` to get frames from other times.
Simple Video Filter	This video filter is found in the SDK folder of the Video Effects in the Effects Control panel. It has a color picker parameter and slider parameter in the Effects Control panel, and modifies the source pixels based on the parameters. If the slider is zero, the filter adds the RGB values in the color picker to the RGB values of each pixel and preserves the alpha. If the slider is non-zero, the filter uses the callback to get the current frame. Using the callback for this purpose is purely for demonstration purposes. The current frame is passed in through `(*theData)->source` and using the callback to get the current frame in a real filter is only wasting time!

Field-Aware Video Filter	This video filter is found in the SDK folder of the Video Effects in the Effects Control panel. It supports 8-bit YUV and RGB. It has a color picker parameter, a slider parameter, and an unused angle parameter in the Effects Control panel, and modifies the source pixels based on the parameters and current field rendering. If the field rendering is upper fields first, it will blend the upper fields of the upper half of the image with the color parameter by the percentage specified by the slider parameter. If the field rendering is lower fields first, it will blend the lower fields of the lower half of the image. If the field rendering is off, it will blend every other row of pixels. The alpha is preserved. It demonstrates use of PPix Suite and Pixel Format Suite. When the setup button is pressed, it displays a message box on Windows, and an alert on Mac OS.
Device	This device controller pretends to control a hardware device. To select it, choose Edit > Preferences > Device Control > Devices: SDK Device Control. When the device control Options button is pressed or Export To Tape is selected, it displays a message box on Windows, and an alert on Mac OS. It reports status in the status pane of the Capture panel, and a simulated timecode location in response to the transport controls. It demonstrates a sample error message when using the Step Back button at time zero.

How To Build the SDK Projects

The required development environment is described in the SDK Audience section.

On Windows, to specify where you want the built plug-ins to go, set a user environment variable (right-click My Computer > Properties > Advanced > Environment Variables > User variables) called PREMSDKBUILDPATH to the desired path (e.g. "C:\Program Files\Adobe\Common\Plug-ins\[version]\MediaCore\"), and re-log in so that the variable will be set. See the screenshot below. In the Visual Studio Property Pages for each SDK project, this path is the base path for the Output File.



Create a new user environment variable to specify the build target directory.

On Mac OS, go to the XCode Preferences, in the Building panel, and set "Place Build Products in:" to "Customized location:". There you can specify the build target folder.

Load 'Em Up!

Plug-in Caching

On its first launch, Premiere Pro loads all the plug-ins, reads the PiPL resource, and sends any startup selectors to determine the plug-ins' capabilities. To save time on future application launches, it saves some of these capabilities in what we call the plug-in cache (the registry on Windows, a Property List file on Mac OS). The next time the application is launched, the cached information is used wherever possible, rather than loading the plug-ins. Caching plug-ins will make the application launch faster, but it may be undesirable for plug-ins that need to be initialized every time. These include plug-ins that need to get run-time information that might change in between app launches (i.e. installed codec lists), and plug-ins that check for hardware and need to be able to fail. So we give your plug-in control final say over whether or not it is reloaded each time.

By default, importers, recorders, exporters, and players are not cached. Exporters can be cached by setting `exExporterInfoRec.isCacheable` to non-zero during *exSelStartup*. The rest can be cached if desired by returning `*IsCacheable` instead of `*NoError` (e.g. for importers, `imIsCacheable` instead of `imNoError`) on the startup selector. On Mac OS, do not set `rmIsCacheable` for recorder plug-ins, as this will cause problems when the recorder is loaded from the cache (bug 1546820).

By default, transitions, video filters, and device controllers are cached by default. To specify that they must be reloaded each time, rather than cached, Premiere effects and transitions should respond to *fsCacheOnLoad* and *esCacheOnLoad*, respectively.

Resolving Plug-in Loading Problems

There are various tools to help in the development process.

On Windows only, you can force Premiere to reload all the plug-ins by holding down shift on startup. The plug-in cache on Mac OS may be deleted manually from the user folder, at `~/Library/Preferences/com.Adobe.Premiere Pro [version].plist`.

For plug-in loading issues, you may first check one of the plug-in loading logs.
On Windows Vista:

```
[user folder]\AppData\Roaming\Adobe\Premiere Pro\[version number]\[process name] Plugin Loading.log
```

On MacOS, this is:

```
~/Library/Application Support/Adobe/Premiere Pro/[version number]/[process name] Plugin Loading.log
```

Your plug-in will be listed by path and filename, and the log will contain details on what happened during the plug-in loading process. If the log says a plug-in has been marked as Ignore, then you should force Premiere to reload all the plug-ins as described above.

No Shortcuts Allowed

The Premiere Pro plug-in loader does not follow Windows shortcuts. Although it does follow Mac OS symbolic links, we recommend against using symbolic links in the plug-ins folder, since the plug-in loader checks the timestamp of the symbolic link rather than the timestamp of the plug-in pointed to. Explanation: If you use a symbolic link and the plug-in fails to load once (for example, if the plug-in pointed to isn't there) it will be marked to ignore when Premiere launches. Even if the plug-in is restored to the proper location, the plug-in loader will check the modification time of the symbolic link, rather than the plug-in pointed to, and continue to ignore the plug-in until

the modification date of the symbolic link is updated. So plug-ins should be placed directly in a plug-ins folder or subfolder.

Debugging Plug-ins

Attaching The Debugger Using Microsoft Visual Studio .NET

Processes may be attached to for debugging after they have been launched. You can do this in Visual Studio via Debug > Process > Attach, or programmatically using `_asm int 3` or `DebugBreak()`. You will then receive the Microsoft error reporting message, but if you hit the Debug button you will enable Just-In-Time Debugging and can attach to the process.

Attaching The Debugger Using XCode

If you attach to one of the Adobe Premiere Pro processes using the unmodified sample projects, you will get the error "No launchable executable present at path." In the Groups & Files panel of XCode, add a New Custom Executable to the Executables group. Specify the full path to the Adobe Premiere Pro app or process you wish to debug. Now you will be able to attach the debugger to the process.

In 4.0.1 and later, there is a new library included that unfortunately has copy protection: /Applications/Adobe Premiere Pro CS4/Adobe Premiere Pro CS4.app/MediaIO/codecs/SurCode.framework
This library will prevent debugging using XCode. To workaround the problem on a development system, temporarily move the library out of the codecs folder.

Dog Ears

Premiere Pro's built-in player has a mode to display statistics, historically known as "dog ears", which can be useful in debugging and tuning performance of importers and effects. The statistics include frames per second, frames dropped during playback, pixel format rendered, render size, and field type being rendered.To use this feature, you'll need to bring up the debug console in Premiere Pro. You can do this via Ctrl/Cmd-F12. To enable the dog ears, type this:

```
debug.set EnableDogEars=true
```

to disable, use this:

```
debug.set EnableDogEars=false
```

If the enter keystroke seems to go to the wrong panel, this is an intermittent panel focus problem. Click the Tools or Info panel before typing in the Console panel, and the enter key will be processed properly.

Once enabled, the player displays the statistics as black text on a partially transparent background. This allows you to still see the underlying video (to some extent) and yet also read the text. When you turn off dog ears, the setting may not take effect until you switch or reopen your current sequence.

Plug-in Installation

Plug-ins must have an installer. This simplifies installation by the user, provides more compact distribution media, and ensures all the pieces are installed correctly. Create a container folder for your plug-in(s) to minimize user confusion. Don't unintentionally overwrite existing plug-ins, or replace newer versions. The installer should find the default installation directories as described below, but should also allow the user to specify an alternate directory.

In Premiere Pro CS3 and later, plug-ins should be installed in the common plug-in location. Supported Premiere and After Effects plug-ins installed here will be loaded by Premiere Pro, After Effects, Encore, SoundBooth, and Media Encoder. Other plug-in types, such as QuickTime and VfW codecs should be installed at the operating system level.

On Windows, the common plug-in path can be found in the registry in the following key:
HKEY_LOCAL_MACHINE/Software/Adobe/Premiere Pro/CurrentVersion/Plug-InsDir

On Mac OS, this common plug-in location is at:
/Library/Application Support/Adobe/Common/Plug-ins/[version]/MediaCore

Presets and editing mode XML files are loaded from the application-specific folders, not from the common location. On Windows, the root path for Premiere Pro is in the registry at HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\Adobe Premiere Pro.exe. Then, just add the proper subdirectories.
For sequence presets:
\Settings\SequencePresets\[Your specific folder]\
Editing modes:
\Settings\Editing Modes\
Sequence preview presets:
\Settings\EncoderPresets\SequencePreview\[Your editing mode GUID]\
Encoder presets:
\MediaIO\systempresets\[Your exporter folder]\
Effects presets:
\Plug-ins\[language subdirectory]\Effect Presets\ (see Localization for the list of language codes)

Plug-in Naming Conventions

On Windows, Premiere Pro plug-ins must have the file extension ".prm". On Mac OS, they have the file extension ".bundle". Other supported plug-in standards use their conventional file extensions: ".aex" for After Effects plug-ins, ".dll" for VST plug-ins.

While it is not required for your plug-in to load, naming your plug-ins using the plug-in type as a prefix (e.g. ImporterSDK, FilterSDK, etc.) will help reduce user confusion.

Creating Sequence Presets

Not to be confused with encoder presets or sequence preview encoder presets, sequence presets are the successor to project presets. They contain the video, audio, timecode, and track layout information used when creating a new sequence.

When providing a new editing mode, provide user-friendly Sequence Presets for the New Sequence dialog. Save the settings with a descriptive name and comment. Emulate our settings files. Install the presets as described in the section, "Plug-in Installation".

Application-level Preferences

For Windows Vista restricted user accounts, the only place that code has guaranteed write access to a folder is inside the user documents folder and its subfolders.
..\Users\[user name]\AppData\Roaming\Adobe\Premiere Pro\[version]\
This means that you cannot save data or documents in the application folder. There is currently no plug-in level API for storing preferences in the application prefs folder. Plug-ins can create their own preferences file in the user's Premiere prefs directory like so:

```
HRESULT herr =
 SHGetFolderPath(NULL, CSIDL_APPDATA, NULL, 0, preferencesPath);
strcat(preferencesPath,
 "\\Adobe\\Premiere Pro\\[version]\\MyPlugin.preferences");
```

Localization

The language used by Premiere Pro is decided by the user during installation. Plug-ins can determine this setting from the following locations:
On Windows, in the registry at HKEY_CURRENT_USER\Software\Adobe\Premiere Pro\[version], in a key named "Language".
On Mac OS, at ~/Library/Preferences/com.Adobe.Premiere Pro.[version].plist, at Root > Language.
The string will be set to one of the values below by Premiere Pro at startup.

Language	String
English	en_US
French	fr_FR
German	de_DE
Italian	it_IT
Japanese	ja_JP
Spanish	es_ES
Korean (new in CS4)	ko_KR

Changing the string will not change the language Premiere Pro runs in, unless you override the application language by placing a file in the following location:

Windows: [App installation folder]\lang-override.txt
Mac OS: [App Installation folder]/[Premiere Pro application package]/Contents/lang-override.txt

Best Practices

When a plug-in receives a selector it doesn't recognize, it should always return the code specific to the plug-in type that means the selector is not supported (i.e. imUnsupported, rmUnsupported, etc). In this way, new selectors can be added to the API and legacy plug-ins will automatically answer whether or not they support it.

Structure Alignment

All the sample projects include PrSDKTypes.h. This header sets the proper (single-byte) structure alignment and specifies the necessary (C-style) external linkage.

Getting Support and Providing Feedback

Please read relevant sections of this document and view the included sample code before requesting assistance. Please direct questions regarding installation, configuration, or use of Adobe products to Adobe Technical Support.

To report a bug or submit a feature request for Premiere Pro, please visit:
http://www.adobe.com/cfusion/mmform/index.cfm?name=wishform
Since this form can be used to submit bugs and features for all Adobe products, make sure you have the right product selected.

We encourage you to use the Premiere Pro SDK forum to ask questions on the API and general integration. For development questions that you'd rather keep confidential, you may contact API Engineering directly. Your feedback can improve the API and SDK to streamline future development.

2 Resources

There are two types of special resources that are specific to Premiere plug-ins: the PiPL and the IMPT. This chapter describes these resources, and how certain plug-in types use them.

Plug-In Property Lists (PiPL) Resource

For many plug-in types, Premiere loads a PiPL (Plug-in Property List) resource. The PiPL is described in a file with a ".r" extension. The complete PiPL syntax is described in PiPL.r. You'll notice that PiPLs are really old. A vestige of 68k Mac OS programming, they spread to Windows. However, if you develop from the sample projects, you shouldn't have to do anything to get them to build properly for Latin languages.

Which Types of Plug-ins Need PiPLs?

Exporters, players, and recorders do not need PiPLs.

Standard importers do not need PiPLs. Synthetic and custom importers use a basic PiPL to specify their name, and the match name that Premiere uses to identify them. The name appears in the File > New menu.

Device controllers use a basic PiPL to specify their name and the match name that Premiere uses to identify them.

Video filters use an extended PiPL to specify their name, the match name that Premiere uses to identify them, the bin they go in, how they handle pixel aspect ratio, whether or not they have randomness, and their parameters. For more information on the ANIM_FilterInfo and ANIM_ParamAtom sections, see the resources section in the Video Filters chapter.

Transitions use an extended PiPL to specify their name, the match name that Premiere uses to identify them, the bin they go in, their description that appears in the Effect Controls Panel, the directional arrow controls, whether or not they have a setup button, whether or not they receive an esSetup when the transition is applied, whether or not they are treated as a transition [TODO

- Is this used anymore?], whether or not they have a Reverse checkbox, whether or not they have borders, the point controls, and how they handle pixel aspect ratio. For more information on the Pr_Effect_Info section, see the resources section in the Transitions chapter.

A Basic PiPL Example

```
#define plugInName          "SDK Custom Import"
#define plugInMatchName     "SDK Custom Import"

resource 'PiPL' (16000) {
{
      // The plug-in type
      Kind {PrImporter},

      // The name as it will appear in a Premiere menu, this can be localized
      Name {plugInName},

      // The internal name of this plug-in - do not localize this. This is used for both Premiere
            and After Effects plug-ins.
      AE_Effect_Match_Name {plugInMatchName}

// Transitions and video filters define more PiPL attributes here

}
};
```

How PiPLs Are Processed By Resource Compilers

On Mac OS, .r files are processed natively by XCode, as a Build ResourcesManager Resources Build Phase. This step is already set for the sample projects.

On Windows, .r files are processed with CnvtPiPL.exe, which creates an .rcp file based upon custom build steps in the project. The .rcp file is then included in the .rc file along with any other resources the plug-in uses. These custom build steps are already in place in the sample projects. To view them, open up the sample project in .NET. In the Solution Explorer, right-click the .r file and choose Properties. In the dialog, choose the Custom Build Step folder. The Command Line contains the script for executing the CnvtPiPL.exe. Unless you are using a different compiler than the support compiler, or adding support for Asian languages, you should not need to modify the custom build steps. This script may also be found as a text file in the SDK at \Examples\Resources\Win\Custom Build Steps.txt. This text file also describes the additional switches used for Asian languages.

IMPT Resource

Premiere Pro looks for an IMPT resource to identify a plug-in as an importer. Before Premiere Pro 1.0, the IMPT resource was also used to declare the file extension supported by an importer. Since file extensions are now declared during imGetIndFormat, the drawtype fourcc in the IMPT resource is no longer used by Premiere Pro. However, a unique drawtype fourcc is needed for the importer to function properly in After Effects on Mac OS. Do not use 0x4D4F6F76. This is already reserved by After Effects.

```
1000 IMPT DISCARDABLE
BEGIN
     0x12345678 // Put your own unique hexadecimal code here
END
```

3 Universals

This chapter covers topics that are common to more than one type of Premiere plug-in. We start by discussing fundamental concepts and common data structures. The rest of the chapter discusses the various function suites that are available to plug-ins.

Time

There are two different representations of time: scale over sampleSize, and ticks.

scale over sampleSize

The first representation of time uses `value/scale/sampleSize` components, either separated, or combined in a `TDB_TimeRecord` structure. `scale` over `sampleSize` defines the timebase. For example, to represent the NTSC standard of 29.97 frames per second, `scale` = 30000 and `sampleSize` = 1001. To represent the PAL standard of 25 frames per second, `scale` = 25 and `sampleSize` = 1. To represent the 24P standard of 23.976, `scale` = 23976 and `sampleSize` = 1000. To represent most other timebases, use `sampleSize` = 1, and `scale` is the frame rate (e.g. 15, 24, 30 fps, etc). Another way of thinking about `scale` and `sampleSize` is that `sampleSize` is the duration of a frame of video, and `scale` is that duration of a second of video.

`value` is the time in the timebase given by `scale` over `sampleSize`. So, for example, 30 frames with a sampleSize of 1001 have a value of 30030. To convert `value` to seconds, divide by `scale`. To convert `value` to frames, divide by `sampleSize`.

Sometimes, as when handling audio-only media, `sampleSize` refers to a sample of audio, and `sampleSize` = 1. In this case, `scale` is the audio sampling rate (22050, 32000, 44100, 48000 Hz, etc).

PrTime

Most newer areas of the API use a tick-based time value that is stored in a signed 64-bit integer. Variables that use this new format are of type `PrTime`. When a frame rate is represented as a PrTime, the frame rate is the number of ticks in a frame duration.

The current number of ticks per second must be retrieved using the callback in the Time Suite. This rate is guaranteed to be constant for the duration of the application's run-time.

Video Frames

Premiere stores each video frame in a `PPix` structure. A `PPixHand` is a handle to a `PPix`. This structure should not be accessed directly, but manipulated using various suites such as the PPix Suite, PPix 2 Suite, PPix Creator Suite, and PPix Creator 2 Suite. Yes, our engineers have had to buy replacement 'P' keys for their keyboards.

Far from being just a boring buffer of RGB data, `PPix`es contain a significant amount of information about a video frame, including: rectangle bounds (width, height), pixel aspect ratio, pixel format, field order, and more.

In the pixel buffer itself, there may be padding between neighboring horizontal rows of pixels. So when iterating through the pixels in the buffer, don't assume that the first pixel on the next line is stored immediately after the last pixel on the current line. Honor the rowbytes, which is a measure of the size in bytes of a row of pixels, including any extra padding.

Frames are guaranteed to be 16-byte aligned.

Pixel Formats and Color Spaces

As of CS5, Premiere supports 54 different pixel formats, not including raw and custom formats. Why so many? Each pixel format has it's unique advantages and disadvantages. 8-bit formats are compact, but lack quality. 32-bit ones are more accurate, but overkill in some situations. Compressed formats are great for storing raw frames, but bad for effects processing. And so on... The conclusion - choose wisely!

What Format Should I Use?

Starting in CS4, plug-ins no longer need to support 8-bit BGRA at a minimum. If required, Premiere can make intermediate format conversions in the render pipeline, although these intermediate conversions will be avoided if possible. Previously in CS3 and earlier, all plug-ins except importers needed to support 8-bit per channel BGRA, even if they supported other formats.

When choosing which pixel formats to support, there are different factors to consider, depending on the plug-in type.

Importers

Importers should provide frames in a format closest to the source format. If needed, Premiere can convert any compressed format to a 8-bit or 32-bit uncompressed format. Keeping the format compressed as long as possible as it passes through the render pipeline will save memory and bandwidth.

Effects

Effects should support the uncompressed format(s) that works best with the effect's pixel processing algorithm. If the algorithm is based on RGB pixel calculations, provide a fast render path using 8-bit BGRA, and optionally a high-quality render path using 32-bit BGRA. If the algorithm is Y'UV-based, use the VUYA pixel formats.

Exporters and Players

Exporters and players should request frames in a format closest to the output format. New in CS5, `PrPixelFormat_Any` can be used in render requests. Any render function that takes a list of pixel formats can now be called with just two formats - the desired 4:4:4:4 pixel format, and `PrPixelFormat_Any`. This allows the host to avoid frame conversions and decompressions in many very common cases. The best part is that the plug-in doesn't need to understand all the possible pixel formats to make use of this. It can use the Image Processing Suite to copy/convert from any a PPix of any format to a separate memory buffer, which is a copy that would likely need to be done anyway.

After the request is made, Premiere analyzes the preferred format of all importers and effects that are used to produce a single rendered frame, as well as the list of requested formats, and chooses the best format to use on a per-segment basis. If the requestor supports more than one format, and the importers and effects used for various clips in the sequence support different formats, the render may use different formats for each segment.

Premiere Pro's built-in Rec. 601 to 709 color space conversion can be slow. So if the majority of the sources and effects use the Rec 601 color space, and if the exporter or player can handle the 601 to 709 conversion quickly on its own, it may be faster to do the color space conversion in the exporter or player.

Other Considerations

For high-bit depth support, the 32f formats are the recommended route, rather than the 16u formats. For example, an exporter that supports 10-bit Y'UV should ask for frames in 32f Y'UV format, and then convert the 32f to 10u.

The ARGB formats can be natively used in the After Effects render pipeline, and are used by After Effects effect plug-ins that do not specifically support any other pixel format. However, in Premiere Pro, these ARGB formats will require byte-swapping, and shouldn't be used.

Byte Order

BGRA, ARGB, and VUYA are written in order of increasing memory address from left to right. Uncompressed formats have a lower-left origin, meaning the first pixel in the buffer describes the pixel in the lower-left corner of the image. Compressed formats have format-specific origins.

PrPixelFormat	Bits / Channel	Format	Additional Details
Unpacked, Uncompressed			
BGRA_4444_8u	8	RGB	
VUYA_4444_8u	8	Y'UV	
VUYA_4444_8u_709	8	Y'UV	Rec. 709 color space. New in Premiere Pro 4.1.
ARGB_4444_8u	8	RGB	For After Effects support. For Premiere Pro, use BGRA.
BGRX_4444_8u	8	RGB	Implicitly opaque alpha channel. The actual data may be left filled with garbage, which allows you to optimize your processing, with the understanding the the alpha channel is opaque. New in Premiere Pro CS5.
VUYX_4444_8u	8	Y'UV	
VUYX_4444_8u_709	8	Y'UV	
XRGB_4444_8u	8	RGB	
BGRP_4444_8u	8	RGB	Premultiplied alpha. New in Premiere Pro CS5.
VUYP_4444_8u	8	Y'UV	
VUYP_4444_8u_709	8	Y'UV	
PRGB_4444_8u	8	RGB	
BGRA_4444_16u	16	RGB	
ARGB_4444_16u	16	RGB	For After Effects support. For Premiere Pro, use BGRA.

PrPixelFormat	Bits / Channel	Format	Additional Details
BGRX_4444_16u	16	RGB	Implicitly opaque alpha. New in Premiere Pro CS5.
XRGB_4444_16u	16	RGB	
BGRP_4444_16u	16	RGB	Premultiplied alpha. New in Premiere Pro CS5.
PRGB_4444_16u	16	RGB	
BGRA_4444_32f	32	RGB	
VUYA_4444_32f	32	Y'UV	
VUYA_4444_32f_709	32	Y'UV	Rec. 709 color space. New in Premiere Pro 4.1.
ARGB_4444_32f	32	RGB	For After Effects support. For Premiere Pro, use BGRA.
BGRX_4444_32f	32	RGB	Implicitly opaque alpha. New in Premiere Pro CS5.
VUYX_4444_32f	32	Y'UV	
VUYX_4444_32f_709	32	Y'UV	
XRGB_4444_32f	32	RGB	
BGRP_4444_32f	32	RGB	Premultiplied alpha. New in Premiere Pro CS5.
VUYP_4444_32f	32	Y'UV	
VUYP_4444_32f_709	32	Y'UV	
PRGB_4444_32f	32	RGB	
Packed, Uncompressed			
YUYV_422_8u_601	8	YUY2	New in Premiere Pro CS4.
YUYV_422_8u_709	8	YUY2	Rec. 709 color space. New in Premiere Pro CS4.
UYVY_422_8u_601	8	UYVY	New in Premiere Pro CS4.
UYVY_422_8u_709	8	UYVY	Rec. 709 color space. New in Premiere Pro CS4.
V210_422_10u_601	10	V210	New in Premiere Pro CS4.
V210_422_10u_709	10	V210	Rec. 709 color space. New in Premiere Pro CS4.
Linear			
BGRA_4444_32f_Linear	32	RGB	These RGB formats have a gamma of 1, rather than the standard 2.2. New in Premiere Pro CS5.
BGRP_4444_32f_Linear	32	RGB	
BGRX_4444_32f_Linear	32	RGB	
ARGB_4444_32f_Linear	32	RGB	
PRGB_4444_32f_Linear	32	RGB	
XRGB_4444_32f_Linear	32	RGB	
Compressed			

PrPixelFormat	Bits / Channel	Format	Additional Details
NTSCDV25	8	DV25	
PALDV25	8	DV25	
NTSCDV50	8	DV50	New in Premiere Pro 3.1.
PALDV50	8	DV50	New in Premiere Pro 3.1.
NTSCDV100_720p	8	DV100 720p	New in Premiere Pro 3.1.
PALDV100_720p	8	DV100 720p	New in Premiere Pro 3.1.
NTSCDV100_1080i	8	DV100 1080i	New in Premiere Pro 3.1.
PALDV100_1080i	8	DV100 1080i	New in Premiere Pro 3.1.
YUV_420_MPEG2_FRAME_PICTURE_PLANAR_8u_601	12	Y'UV 4:2:0	Progressive Rec. 601 color space
YUV_420_MPEG2_FIELD_PICTURE_PLANAR_8u_601	12	Y'UV 4:2:0	Interlaced Rec. 601 color space
YUV_420_MPEG2_FRAME_PICTURE_PLANAR_8u_709	12	Y'UV 4:2:0	Progressive Rec. 709 color space
YUV_420_MPEG2_FIELD_PICTURE_PLANAR_8u_709	12	Y'UV 4:2:0	Interlaced Rec. 709 color space
Raw	?	?	Raw, opaque data, with no rowbytes or height

Custom Pixel Formats

New in CS4, custom pixel formats are supported. Plug-ins can define a pixel format which can pass through various aspects of our pipeline, but remain completely opaque to the MediaCore renderer. Use the macro `MAKE_THIRD_PARTY_CUSTOM_PIXEL_FORMAT_FOURCC` in the Pixel Format Suite. Please use a unique name to avoid collisions. The format doesn't need to be registered in any sense. They can just be used in the same way the current pixel formats are used, though in many cases they will be ignored.

The first place the new pixel formats can appear in the render pipeline is at the importer level. Importers can advertise the availability of these pixel formats during *imGetIndPixelFormat*, just as they would for any other format. Note that importers must also support a non-custom pixel format, for the case where the built-in renderer is used, which would not be prepared to handle an opaque pixel format added by a third-party. In the importer, use the new `CreateCustomPPix` call in the PPix Creator 2 Suite, and specify a custom pixel format and a memory buffer size, and the call will pass back a `PPix` of the requested format. These PPixes can then be returned from an importer, like any other. The memory for the `PPix` will be allocated by MediaCore, and must

be a flat data structure as they will need to be copied between processes. However, because the data itself is completely opaque, it can easily be a reference to another pixel buffer, as long as the reference can be copied. For example, the buffer could be a constant 16 bytes, containing a GUID which can be used to access a memory buffer by name in another process.

To query for available custom pixel formats from the player, use the `GetNumCustomPixelFormats` and `GetCustomPixelFormat` calls in the Clip Render Suite. The custom pixel formats will not returned by the regular calls to get the supported frame formats, mostly to prevent them from being used. The other Clip Render Suite functions will accept requests for custom pixel formats and will return these custom PPixes like any others. With the Clip Render Suite, a third-party player can directly access these custom `PPix`es from a matched importer.

Smart Rendering

Smart rendering involves passing compressed frames from the importer to the exporter, to bypass any unnecessary decompression and recompression, which reduces quality and performance. The way to implement this is by passing custom PPixes between an importer, exporter, and usually a renderer.

In the rare case of exporting a single clip, using the Clip Render Suite in the exporter to request custom PPixes from the importer is sufficient. But in the more common case of exporting a sequence, a renderer that supports the custom pixel format is required.

When an exporter running in Media Encoder parses the segments in the sequence, it only has a very high-level view. It sees the entire sequence as a single clip (which is actually a temporary project file that has been opened using a Dynamic Link to the PProHeadless process), and it sees any optional cropping or filters as applied effects. So when the exporter parses that simple, high-level sequence, if there are no effects, it should use the MediaNode's `ClipID` with the Clip Render Suite to get frames directly from the PProHeadless process. In the PProHeadless process, the renderer can step in and parse the real sequence in all its glory. It can use the Clip Render Suite to get the frames in the custom pixel format directly from the importer, and then set the custom `PPix` as the render result. This custom `PPix` then is available to the exporter, in a pristine, compressed `PPix`.

Pixel Aspect Ratio

Pixel Aspect Ratio (PAR) is usually represented as a rational number, with a numerator and a denominator. Note that several PAR values were changed in CS4 to match broadcast standards. Here are some examples of pixel aspect ratios:

NTSC DV 0.9091 PAR is (10, 11)
NTSC DV Widescreen 1.2121 PAR is (40, 33)

PAL DV 1.0940 PAR is (768, 702)
PAL DV 1.4587 PAR is (1024, 702)
Square 1.0 PAR is (1,1)

In certain legacy structures, PAR is represented as a single long, such as in `recCapInfoRec.pixelAspectRatio`. This uses a representation where the numerator is bit-shifted 16 to the left, and OR'd with the denominator. For example NTSC DV 0.9091 PAR is `(10 << 16) | 11`.

Fields

There are different constants defined for fields. These constants are now interchangable in CS4, since the conflicting constants for the old compiler API have been removed.

Recorders	Exporters, Players, getSettings(), etc
`kMALFieldsNone`	`prFieldsNone`
`kMALFieldsUpperFirst`	`prFieldsUpperFirst`
`kMALFieldsLowerFirst`	`prFieldsLowerFirst`
`kMALFieldsUnknown`	`prFieldsUnknown`
`kMALFieldsInvalid`	

Audio

32-bit Float, Uninterleaved Format

All audio calls to and from Premiere use arrays of buffers of 32-bit floats to pass audio. Audio is not interleaved, rather separate channels are stored in separate buffers. So the structure for stereo audio looks like this:

```
float* audio[2];
```

where `audio[0]` is the address of a buffer N samples long, and `audio[1]` is the address of a second buffer N samples long. `audio[0]` contains the left channel, and `audio[1]` contains the right channel. N is the number of sample frames in the buffer.

Since Premiere uses 32-bit floats for each audio sample, it can represent values above 0 dB. 0 dB corresponds to +/- 1.0 in floating point. A floating point sample can be converted to a 16-bit short integer by multiplying by 32767.0 and casting the result to a short. E.g.:
`sample16bit[n] = (short int) (sample32bit[n] * 32767.0)`

The plug-in is responsible for converting to and from the 32-bit uninterleaved format when reading a file that uses a different format. There are calls to convert between formats in the Audio Suite.

Audio Sample Types

Since 32-bit floats are the only audio format ever passed, there is no option of sample type or bit depth. However, file formats do use a variety of sample types and bit depths, so `AudioSampleTypes` define a variety of possible formats. These formats are used to set members in structures passed to Premiere to define the user interface, and do not affect the format of the audio passed to and from Premiere.

PrAudioSampleType	Description
`kAudioSampleType_8BitInt`	8-bit integer
`kAudioSampleType_8BitTwosInt`	8-bit integer, two's complement
`kAudioSampleType_16BitInt`	16-bit integer
`kAudioSampleType_24BitInt`	24-bit integer
`kAudioSampleType_32BitInt`	32-bit integer
`kAudioSampleType_32BitFloat`	32-bit floating point
`kAudioSampleType_64BitFloat`	64-bit floating point
`kAudioSampleType_16BitIntBigEndian`	16-bit integer, big endian
`kAudioSampleType_24BitIntBigEndian`	24-bit integer, big endian
`kAudioSampleType_32BitIntBigEndian`	32-bit integer, big endian
`kAudioSampleType_32BitFloatBigEndian`	32-bit floating point, big endian
`kAudioSampleType_Compressed`	Any non-PCM format
`kAudioSampleType_Packed`	Any PCM format with mixed sample types
`kAudioSampleType_Other`	A sample type not in this list
`kAudioSampleType_Any`	Any available sample type (used by exporters)

Audio Sample Frames

A sample frame is a unit of measurement for audio. One audio sample frame describes all channels of one sample of audio. Each sample is a 32-bit float. Thus, the storage requirement of an audio sample frame in bytes is equal to 4 * number of channels.

Audio Sample Rate

`PrAudioSample` is a `prInt64`

Audio Channel Types

Premiere currently supports three different audio channel types: mono, stereo, and 5.1. The order of the stereo channels is: left, right. The order of the 5.1 channels is: left, right, left surround, right surround, center, LFE.

New in Premiere Pro 4.0.1, there is partial support for a 16 channel master audio track, only for importing OMFs and playing out to hardware. The 16-channel master audio track should not be used for export.

PrAudioChannelType	Description
`kPrAudioChannelType_Mono`	Mono
`kPrAudioChannelType_Stereo`	Stereo
`kPrAudioChannelType_51`	5.1
`kPrAudioChannelType_16Channel`	16 channel, support for master audio track playback only

Memory Management

Premiere Pro has a media cache in which it stores imported frames, intermediate frames (intermediate stages of a render), fully rendered frames, and audio. This is sized based on a specific percentage of physical memory, taking into account if multiple Production Premium applications like After Effects, Encore, etc are also running. PPro manages this cache itself, so as it adds new items to the cache, it flushes least recently used items.

What Really is a Memory Problem?

Often, users monitoring memory usage are alarmed when they see memory growing to a specific point during a render or playback. When the memory doesn't drop right back down after a render or playback, they might think they have found a memory leak. However, keeping in mind the function of the Premiere Pro media cache, this behavior is to be expected.

On the other hand, memory contention between plug-ins and the rest of Premiere Pro can lead to memory problems. If a plug-in allocates a significant amount of memory and the Premiere Pro media cache has not accounted for it, this means there is less free memory available after the media cache grows to the predefined size. Even if Premiere Pro does not completely run out of

memory, limited memory can cause memory thrashing as memory is moved around to make room for video frames, which in turn can cause poor performance.

Solutions for Memory Contention

The best approach to reduce memory contention is to reduce the memory requirements of each plug-in. However, if the memory requirements of a plug-in are significant, it should also use the Memory Manager Suite to report any memory usage that would not already be accounted for. Frames allocated using the PPix Creator Suite are accounted for, but any memory allocated using the old PPix and Memory functions are not automatically accounted for.

If each instance of an importer has very high memory requirements, importers can set the flag `imFileInfoRec8.highMemUsage` = kPrTrue during *imGetInfo8*. This makes Premiere Pro limit the number of open file instances with the flag set to true.

Basic Types and Structures

These types and structures are defined in PrSDKTypes.h and PrSDKStructs.h, and are used throughout the Premiere API. Premiere defines cross-platform types for convenience when developing plug-ins for both Windows and Mac OS.

Name	Description
`prColor`	An unsigned 32-bit integer that stores an RGB color. This type is useful for the 8-bpc colors retrieved by the color picker in a video effect or transition. Color channels are stored as BGRA, in order of increasing memory address from left to right.
`prWnd`	A Windows `HWND` or Mac OS `NSView*`
`prOffscreen`	A Windows `HDC`
`prRect`	A Windows `RECT` or Mac OS `Rect`. Use the utility function `prSetRect` to set the dimensions of a `prRect` struct. This should be used because Mac OS `Rect` members have a different ordering than Windows `RECT` members.

`prFloatRect`	`typedef struct {`
	`  float left;`
	`  float top;`
	`  float right;`
	`  float bottom;`
	`} prFloatRect;`

`prRgn`	A Windows `HRGN`

Name	Description
prPoint, LongPoint	typedef struct { csSDK_int32 x; csSDK_int32 y; } prPoint, LongPoint; LongPoint is deprecated, but still used for a couple of Bottleneck callbacks
prFPoint	typedef struct { double x; double y; } prFPoint64;
prPixel	(Deprecated)
prPixelAspectRatio	(Deprecated)
PPix, *PPixPtr, **PPixHand	Holds a video frame or field, and contains related attributes such as pixel aspect ratio and pixel format. Manipulate PPixs using the PPix Suite, never directly.
TDB_TimeRecord	A time database record representing a time value in the context of a video frame rate. typedef struct { TDB_Time value; TDB_TimeScale scale; TDB_SampSize sampleSize; } TDB_TimeRecord;
prBool	Can be either kPrTrue or kPrFalse
PrMemoryPtr, *PrMemoryHandle	A char*
PrTimelineID, PrClipID	A 32-bit signed integer.
prUTF8Char	An 8-bit unsigned integer.
PrSDKString	An opaque data type that should be accessed using the new String Suite.

Name	Description

PrParam	Used for exporter parameters
	struct PrParam
	{
	PrParamType mType;
	union
	{
	csSDK_int8 mInt8;
	csSDK_int16 mInt16;
	csSDK_int32 mInt32;
	csSDK_int64 mInt64;
	float mFloat32;
	double mFloat64;
	csSDK_uint8 mBool;
	prFPoint64 mPoint;
	prPluginID mGuid;
	PrMemoryPtr mMemoryPtr;
	};
	};
	
	enum PrParamType
	{
	kPrParamType_Int8 = 1,
	kPrParamType_Int16,
	kPrParamType_Int32,
	kPrParamType_Int64,
	kPrParamType_Float32,
	kPrParamType_Float64,
	kPrParamType_Bool,
	kPrParamType_Point,
	kPrParamType_Guid,
	kPrParamType_PrMemoryPtr
	};

prDateStamp	Used in by importers in imFileAttributesRec.creationDateStamp. typedef struct { csSDK_int32 day; csSDK_int32 month; csSDK_int32 year; csSDK_int32 hours; csSDK_int32 minutes; double seconds; } prDateStamp;

Suites

There are different sets of function suites available to Premiere plug-ins. The SweetPea Suites are the more modern suites that have been added for most new functionality. The piSuites are still needed for various functionality that has not all been superceded by the SweetPea Suites. Whenever possible, use the SweetPea Suites.

There are also function suites more specific to certain plug-in types. The Bottleneck Functions are useful for transitions and video filters. Other suites available to only one plug-in type are documented in the appropriate chapter for that plug-in type.

SweetPea Suites

Overview

Suites common to more than one plug-in type are documented in this chapter below. Suites that are only used by one plug-in type are documented in the chapter on that plug-in type. Below is a table of all suites available in Premiere Pro:

Suite Name	Relevant to Plug-in Type
Accelerated Render Invocation Suite	Exporters, Players, Renderers
App Info Suite	All
Application Settings Suite	All
Async File Reader Suite	Importers
Async Operation Suite	All
Audio Suite	Importers, Exporters
Clip Render Suite	Exporters, Players, Renderers
Deferred Processing Suite	Importers
Error Suite	All
Export File Suite	Exporters
Export Info Suite	Exporters
Export Param Suite	Exporters
Export Progress Suite	Exporters
File Registration Suite	Importers, Transitions, Video Filters
Flash Cue Marker Data Suite	Exporters
Image Processing Suite	All
Importer File Manager Suite	Importers
Legacy Suite	All

Marker Suite	Exporters
Media Accelerator Suite	Importers
Memory Manager Suite	All
Palette Suite	Exporters
Pixel Format Suite	All
Playmod Audio Suite	Players
Playmod Device Control Suite	Players
Playmod Render Suite	Players
PPix Cache Suite	Importers, Players, Renderers
PPix Creator Suite	All
PPix Creator 2 Suite	All
PPix Suite	All
PPix 2 Suite	All
Quality Suite	Players, Renderers
RollCrawl Suite	Exporters, Players, Renderers
Scope Render Suite	Players
Sequence Audio Suite	Exporters
Sequence Info Suite	Importers, Transitions, Video Filters
Sequence Render Suite	Exporters, Renderers
Stock Image Suite	Players
String Suite	All
Threaded Work Suite	Renderers
Time Suite	Players
Video Segment Render Suite	Exporters, Players, Renderers
Video Segment Suite	Exporters, Players, Renderers
Window Suite	All

All SweetPea suites are accessed through the Utilities Suite. Plug-ins can acquire the suites like so:

```
SPBasicSuite *SPBasic = NULL;
PrSDKPixelFormatSuite *PixelFormatSuite = NULL;
SPBasic = stdParmsP->piSuites->utilFuncs->getSPBasicSuite();
if (SPBasic)
{
SPBasic->AcquireSuite (  kPrSDKPixelFormatSuite,
                         kPrSDKPixelFormatSuiteVersion,
                         (const void**)&PixelFormatSuite);
}
```

If for some reason your code depends on a specific older version of a suite, rather than requesting `kPrSDKPixelFormatSuiteVersion`, you request a specific version number instead.

Don't forget to release the suites when finished!

```
if (SPBasic && PixelFormatSuite)
{
SPBasic->ReleaseSuite (  kPrSDKPixelFormatSuite,
                         kPrSDKPixelFormatSuiteVersion);
}
```

App Info Suite

New in CS3. For plug-in types that are shared between different applications, such as After Effects plug-ins, Premiere exporters, players, and importers, it may be important to know which host the plug-in is currently running in. This suite currently provides the host application and version number. See PrSDKAppInfoSuite.h.

Application Settings Suite

New in CS4. This suite provides calls to get the scratch disk folder paths defined in the current project, where the captured files and preview files are created. It also provides a call to get the project file path. All paths are passed back as PrSDKStrings. Use the new String Suite to extract the strings to UTF-8 or UTF-16. See PrSDKApplicationSettingsSuite.h.

Audio Suite

Calls to convert to and from the native audio format used by the Premiere API, at various bit depths. See PrSDKAudioSuite.h.

Clip Render Suite

New in 2.0. Use this suite in the player or renderer, to request source frames directly from the importer. There are calls to find the supported frame sizes and pixel formats, so that the caller can make an informed decision about what format to request. Frames can be retrieved synchronously or asynchronously. Asynchronous requests can be cancelled, for example if the frames have passed their window of playback. See PrSDKClipRenderSuite.h.

Starting in CS4, this suite includes calls to find any custom pixel format supported by a clip, and to get frames in those custom pixel formats.

An exporter can use this suite to request frames from the renderer in a compressed pixel format.

Error Suite

Uses a single callback for errors, warnings, and info. This callback will activate a flashing icon in the lower left-hand corner of the main application window, which when clicked, will open up the new Events Window containing the error information. See PrSDKErrorSuite.h.

Starting in version 3 of the suite, introduced in CS4, the suite supports UTF-16 strings.

File Registration Suite

Used for registering external files (such as textures, logos, etc) that are used by a plug-in instance but do not appear as footage in the Project Window. Registered files will be taken into account when trimming or copying a project using the Project Manager. See PrSDKFileRegistrationSuite.h.

Flash Cue Marker Data Suite

New in CS4. Specific utilities to read Flash cue points. Use in conjunction with the Marker Suite. See PrSDKFlashCueMarkerDataSuite.h.

Image Processing Suite

New in CS5. Various calls to get information on pixel formats and process frames. The `ScaleConvert()` call is the way to copy-convert from a buffer of any supported pixel format to a separate memory buffer.

Marker Suite

New in CS4. New way to read markers of all types. See PrSDKMarkerSuite.h.

Memory Manager Suite

New in Premiere Pro 2.0. Calls to allocate and deallocate memory, and to reserve an amount of memory so that it is not used by the host. See PrSDKMemoryManagerSuite.h.

ReserveMemory

A plug-in instance can call `ReserveMemory` as a request to reserve space so that Premiere's media cache does not use it. Each time `ReserveMemory` is called, it updates Premiere Pro as to how many bytes the plug-in instance is currently reserving. The amount specified is absolute, rather than cumulative. So to release any reserved memory to be made available to Premiere Pro's media cache, call it with a size of 0.

`ReserveMemory` changes the maximum size of Premiere's Media Cache. So if the cache size starts at 512 MB, and you reserve 100 MB, then the cache will not grow beyond 412 MB. `ReserveMemory` will reserve a different amount of memory, depending on the amount of available memory in the system, and what other plug-in instances have already reserved. The media cache needs a minimum amount of memory to play audio, render, etc.

Starting in version 2 of the suite, introduced in CS4, there are calls to allocate/deallocate memory. This is necessary for exporters, which are not passed the legacy `memFuncs`.

Pixel Format Suite

See the table of supported pixel formats. `GetBlackForPixelFormat` returns the minimum (black) value for a given pixel format. `GetWhiteForPixelFormat` returns the maximum (white) value for a given pixel format. Pixel types like YUYV actually contain a group of two pixels to specify a color completely, so the data size returned in this case will be 4 bytes (rather than 2). This call does not support MPEG-2 planar formats.

`ConvertColorToPixelFormattedData` converts an BGRA/ARGB value into a value of a different pixel type. These functions are not meant to convert entire frames from one colorspace to another, but may be used to convert a single color value from a filter color picker or transition border. To convert frames between pixel formats, see the Image Processing Suite.

New in Premiere Pro 4.0.1, `MAKE_THIRD_PARTY_CUSTOM_PIXEL_FORMAT_FOURCC()` defines a custom pixel format.

PPix Cache Suite

Used by an importer, player, or renderer to take advantage of the host application's PPix cache. See PrSDKPPixCacheSuite.h.

Starting in version 2 of this suite, introduced in Premiere Pro 4.1, `AddFrameToCache` and `GetFrameFromCache` now have two extra parameters, `inPreferences` and `inPreferencesLength`. Now frames are differentiated within the cache, based on the importer preferences, so when the preferences change, the host will not use the old frame when it gets a frame request.

PPix Creator Suite

Includes callbacks to create and copy PPixs. See also the PPix Creator 2 Suite.

CreatePPix

Creates a new PPix. The advantage of using this callback is that frames allocated are accounted for in the media cache, and are 16-byte aligned. `ppixNew` and `newPtr` don't allocate memory in the media cache, or perform any alignment.

```
prSuiteError (*CreatePPix)(
     PPixHand*           outPPixHand,
     PrPPixBufferAccess  inRequestedAccess,
     PrPixelFormat       inPixelFormat,
     const prRect*       inBoundingRect);
```

Parameter	Description
`PPixHand *outPPixHand`	The new `PPix` handle if the creation was successful. NULL otherwise.
`PrPPixBufferAccess inRequestedAccess`	Requested pixel access. Read-only is not allowed (doesn't make sense). `PrPPixBufferAccess` values are defined in PPix Suite.
`PrPixelFormat inPixelFormat`	The pixel format of this `PPix`

ClonePPix

Clones an existing `PPix`. It will ref-count the `PPix` if only read access is requested and the `PPix` to copy from is read-only as well, otherwise it will create a new one and copy.

```
prSuiteError (*ClonePPix)(
     PPixHand            inPPixToClone,
     PPixHand*           outPPixHand,
     PrPPixBufferAccess  inRequestedAccess);
```

Parameter	Description
`PPixHand inPPixToClone`	The `PPix` to clone from.
`PPixHand *outPPixHand`	The new `PPix` handle if the creation was successful. NULL otherwise.

Parameter	Description
`PrPPixBufferAccess inRequestedAccess`	Requested pixel access. Only read-only is allowed right now. `PrPPixBufferAccess` values are defined in PPix Suite.

PPix Creator 2 Suite

More callbacks to create PPixs, including raw PPixs. Starting in version 2 of this suite, introduced in Premiere Pro 4.0.1, there is a new `CreateCustomPPix` call to create a PPix in a custom pixel format. See PrSDKPPixCreator2Suite.h.

PPix Suite

Callbacks and enums pertaining to PPixs. See also PPix 2 Suite.

PrPPixBufferAccess

Can be either `PrPPixBufferAccess_ReadOnly`, `PrPPixBufferAccess_WriteOnly`, or `PrPPixBufferAccess_ReadWrite`.

Dispose

This will free this PPix. The PPix is no longer valid after this function is called.

```
prSuiteError (*Dispose)(
     PPixHand inPPixHand);
```

Parameter	Description
`PPixHand inPPixHand`	The `PPix` handle to dispose.

GetPixels

This will return a pointer to the pixel buffer.

```
prSuiteError (*GetPixels)(
     PPixHand            inPPixHand,
     PrPPixBufferAccess  inRequestedAccess,
     char**              outPixelAddress);
```

Parameter	Description
PPixHand inPPixHand	The PPix handle to operate on.
PrPPixBufferAccess inRequestedAccess	Requested pixel access. Most PPixs do not support write access modes.
char** outPixelAddress	The output pixel buffer address. May be NULL if the requested pixel access is not supported.

GetBounds

This will return the bounding rect.

```
prSuiteError (*GetBounds)(
     PPixHand  inPPixHand,
     prRect*   inoutBoundingRect);
```

Parameter	Description
PPixHand inPPixHand	The PPix handle to operate on.
prRect* inoutBoundingRect	The address of a bounding rect to be filled in.

GetRowBytes

This will return the row bytes of the PPix.

```
prSuiteError (*GetRowBytes)(
     PPixHand       inPPixHand,
     csSDK_int32*   outRowBytes);
```

Parameter	Description
PPixHand inPPixHand	The PPix handle to operate on.
csSDK_int32* outRowBytes	Returns how many bytes must be added to the pixel buffer address to get to the next line.

GetPixelAspectRatio

This will return the pixel aspect ratio of this PPix.

```
prSuiteError (*GetPixelAspectRatio)(
     PPixHand       inPPixHand,
     csSDK_uint32*  outPixelAspectRatioNumerator,
```

```
     csSDK_uint32*  outPixelAspectRatioDenominator);
```

Parameter	Description
`PPixHand inPPixHand`	The PPix handle to operate on.
`PrPixelFormat* outPixelFormat`	Returns the pixel format of this PPix

GetUniqueKey

This will return the unique key for this `PPix`. Returns error if the buffer size is too small (call `GetUniqueKeySize` to get the correct size). Returns error if the key is not available. Returns success if the key data was filled in.

```
prSuiteError (*GetUniqueKey)(
     PPixHand       inPPixHand,
     unsigned char* inoutKeyBuffer,
     size_t         inKeyBufferSize);
```

Parameter	Description
`PPixHand inPPixHand`	The `PPix` handle to operate on.
`unsigned char* inoutKeyBuffer`	Storage for the key to be returned.
`size_t inKeyBufferSize`	Size of buffer

GetUniqueKeySize

This will return the unique key size. This will not change for the entire run of the application.

```
prSuiteError (*GetUniqueKeySize)(
     size_t* outKeyBufferSize);
```

Parameter	Description
`size_t* outKeyBufferSize`	Returns the size of the `PPix` unique key.

GetRenderTime

This will return the render time for this `PPix`.

```
prSuiteError (*GetRenderTime)(
     PPixHand       inPPixHand,
     csSDK_int32*   outRenderMilliseconds);
```

Parameter	Description
`PPixHand inPPixHand`	The PPix handle to operate on.
`csSDK_int32* outRenderMillisec-onds`	Returns the render time in milliseconds. If the frame was cached, the time will be zero.

PPix 2 Suite

A call to get the size of a PPix. Starting in version 2 of this suite, introduced in CS4, there is a new `GetYUV420PlanarBuffers` call to get buffer offsets and rowbytes of YUV_420_MPEG2 pixel formats. See PrSDKPPix2Suite.h.

Quality Suite

Contains enumerated values for render and playback qualities. Starting in Premiere Pro 4.0.1, there is a new value, `kPrRenderQuality_Max` quality. When the built-in renderer is called with this quality, it uses new higher quality algorithms for image scaling, compositing, and deinterlacing. The render time required is much longer than high quality. Starting in Premiere Pro 4.1, there is a new `kPrPlaybackQuality_Invalid` value. There is also a new set of `PrPlaybackFractionalResolution` enums for use with the fractional resolution feature in the player.

During playback, it's recommended that render requests be made with medium quality. You may ask for high quality when paused, but high quality can be very slow in simple cases, such that it is not recommended for rendering during playback.

Render Quality

```
typedef enum
{
     kPrRenderQuality_Max = 4,
     kPrRenderQuality_High = 3,
     kPrRenderQuality_Medium = 2,
     kPrRenderQuality_Low = 1,
     kPrRenderQuality_Draft = 0,
} PrRenderQuality;
```

Playback Quality

```
typedef enum {
     kPrPlaybackQuality_Invalid = 4,
     kPrPlaybackQuality_High = 3,
```

```
	kPrPlaybackQuality_Draft = 2,
	kPrPlaybackQuality_Auto = 1,
} PrPlaybackQuality;
```

Playback Fractional Resolutions

```
typedef enum {
	kPrPlaybackFractionalResolution_Invalid = 6,
	kPrPlaybackFractionalResolution_Sixteenth = 5,
	kPrPlaybackFractionalResolution_Eighth = 4,
	kPrPlaybackFractionalResolution_Quarter = 3,
	kPrPlaybackFractionalResolution_Half = 2,
	kPrPlaybackFractionalResolution_Full = 1,
} PrPlaybackFractionalResolution;
```

RollCrawl Suite

Used by a player or renderer to obtain the pixels for a roll/crawl. The player or render can then move and composite it using accelerated algorithms or hardware. See PrSDKRollCrawlSuite.h.

Sequence Info Suite

New in CS4. Calls to get the frame size and pixel aspect ratio of a sequence. This is useful for importers, transitions, or video filters, that provide a custom setup dialog with a preview of the video, so that the preview frame can be rendered at the right dimensions. See PrSDKSequenceInfoSuite.h.

String Suite

New in CS4. Calls to allocate, copy, and dispose of `PrSDKString`s. See PrSDKStringSuite.h.

Threaded Work Suite

New in CS4. Calls to register and queue up a threaded work callback for processing on a render thread. If you queue multiple times, it is possible for multiple threads to call your callback. If this is a problem, you'll need to handle this on your end.

Time Suite

A SweetPea suite that includes the following structure, callbacks, and enum:

pmPlayTimebase

Member	Description
`csSDK_uint32 scale`	rate of the timebase
`csSDK_int32 sampleSize`	size of one sample
`csSDK_int32 fileDuration`	number of samples in file

PrVideoFrameRates

Member	Description
`kVideoFrameRate_24Drop`	24000 / 1001
`kVideoFrameRate_24`	24
`kVideoFrameRate_PAL`	25
`kVideoFrameRate_NTSC`	30000 / 1001
`kVideoFrameRate_30`	30
`kVideoFrameRate_PAL_HD`	50
`kVideoFrameRate_NTSC_HD`	60000 / 1001
`kVideoFrameRate_60`	60
`kVideoFrameRate_Max`	0xFFFFFFFF

GetTicksPerSecond

Get the current ticks per second. This is guaranteed to be constant for the duration of the run-time.

```
prSuiteError (*GetTicksPerSecond)(
     PrTime* outTicksPerSec);
```

GetTicksPerVideoFrame

Get the current ticks in a video frame rate. `inVideoFrameRate` may be any of the `PrVideoFrameRates` enum.

```
prSuiteError (*GetTicksPerVideoFrame)(
     PrVideoFrameRates   inVideoFrameRate,
     PrTime*             outTicksPerFrame);
```

GetTicksPerAudioSample

Get the current ticks in an audio sample rate. Returns `kPrTimeSuite_RoundedAudioRate` if the requested audio sample rate is not an even divisor of the base tick count and therefore times in this rate will not be exact. Returns `kPrTimeSuite_Success` if otherwise.

```
prSuiteError (*GetTicksPerAudioSample)(
     float      inSampleRate,
     PrTime*    outTicksPerSample);
```

Video Segment Render Suite

New in CS5. This suite uses the built-in software path for rendering, and supports subtree rendering. This means the plug-in can ask the host to render a part of the segment, and then still handle the rest of the rendering. This is useful if, for example, one of the layers has an effect that the plug-in cannot render itself. The plug-in can have the host render that layer, but then handle the other layers along with the compositing.

Video Segment Suite

New in CS4. This suite provides calls to parse a sequence and get details on video segments. All the queryable node properties are in PrSDKVideoSegmentProperties.h. These properties will be returned as `PrSDKString`s, and should be managed using the String Suite.

When calling `GetSegmentInfo()`, a plug-in can quickly determine if the segment has changed by comparing the new `outHash` with the plug-in's copy for that segment. If it has changed, then you can start digging into the segment nodes.

The basic structure of the video segments is that of a tree structure. There is a Compositor node with n inputs. Each of those inputs is a Clip node, which has one input which is a Media node, and it also has n Operators, which are effects.

So, a simple example, three clips in a stack, the top one with three effects looks like this:

```
Segment
     Compositor Node
          Clip Node
               Media Node (bottom clip)
          Clip Node
               Media Node (middle clip)
          Clip Node
               Media Node (top clip)
               Clip Operators (Blur, Color Corrector, Motion)
```

To get a good idea of the segment structure, try the SDK player, create a sequence using the SDK Editing Mode, and watch the text overlay in the Sequence Monitor as you perform edits.

See PrSDKVideoSegmentSuite.h and PrSDKVideoSegmentProperties.h.

Window Suite

New in CS4. This is the new preferred way to get the handle of the mainframe window, especially for exporters, who don't have access to the legacy piSuites.

Legacy Callback Suites

piSuites

These callbacks are available to all plug-ins, although many of these callbacks are only appropriate for specific plug-in types.

```
typedef struct {
     int piInterfaceVer;
     PlugMemoryFuncsPtr       memFuncs;
     PlugWindowFuncsPtr       windFuncs;
     PlugppixFuncsPtr         ppixFuncs;
     PlugUtilFuncsPtr         utilFuncs;
     PlugTimelineFuncsPtr     timelineFuncs;
} piSuites, *piSuitesPtr;
```

Member	Description
`piInterfaceVer`	API version Premiere Pro CS4 - `PR_PISUITES_VERSION_9` Premiere Pro CS3 - `PR_PISUITES_VERSION_8` Premiere Pro 2.0 - `PR_PISUITES_VERSION_7` Premiere Pro 1.5.1 - `PR_PISUITES_VERSION_6` Premiere Pro 1.5 - `PR_PISUITES_VERSION_5` Premiere Pro 1.0 - `PR_PISUITES_VERSION_4` Premiere 6.x - `PR_PISUITES_VERSION_3` Premiere 5.1 - `PR_PISUITES_VERSION_2` Premiere 5.0 - `PR_PISUITES_VERSION_1`
`memfuncs`	Pointer to memory functions
`windFuncs`	Pointer window functions
`ppixFuncs`	Pointer PPix functions

utilFuncs	Pointer to utility functions. In the utilFuncs, the getSPBasicSuite callback provides access to the "SweetPea" Suites, which are used for most of the newer functions.
timelineFuncs	Pointer to timeline functions

Memory Functions

Memory and handle allocation. Where possible, use the PPix Creator Suite for PPix-specific allocation.

Strings passed to and from Premiere in API structures are always null-terminated C strings.

Function	Description
newPtr	Allocates a block of memory, returns a pointer to the new block. `char* newPtr (csSDK_uint32 size);`
newPtrClear	Equivalent to newPtr, but initializes the memory to 0. `char* newPtrClear (csSDK_uint32 size);`

setPtrSize	Resizes an allocated memory block.
	
	`void setPtrSize (`
	`PrMemoryPtr *ptr,`
	`csSDK_uint32 newsize);`

getPtrSize	Returns size in bytes of an allocated memory block. `csSDK_int32 getPtrSize (char *ptr);`
disposePtr	Frees an allocated memory block. `void disposePtr (char *ptr);`
newHandle	Allocates a block of memory, returning a handle to it. `char** newHandle (csSDK_uint32 size);`
newHandleClear	Equivalent to newHandle, but initializes the memory to 0. `char** newHandleClear (csSDK_uint32 size);`

setHandleSize	Resizes an allocated memory handle.
	
	`csSDK_int16 setHandleSize (`
	`char **PrMemoryHandle,`
	`csSDK_uint32 newsize);`

Function	Description

`getHandleSize`	Returns the size (in bytes) of an allocated block.
	
	`csSDK_int32 getHandleSize (`
	`  char **PrMemoryHandle);`

`disposeHandle`	Disposes of a previously allocated handle. `void disposeHandle (char **PrMemoryHandle);`
`lockHandle` `unlockHandle`	These legacy functions are deprecated and should no longer be used.

Window Functions

Window management routines. Superceded by the Window Suite.

Function	Description
`updateAllWindows`	Updates all windows. Windows only, doesn't work on Mac OS. `void updateAllWindows (void);`
`getMainWnd`	Returns the main application `HWND`. `void getMainWnd (void);`

PPix Functions

Used to manipulate a `PPix`. Superceded by the PPix Creator Suite for PPix allocation and the PPix Suite for general PPix functions.

Function	Description
`ppixGetPixels`	Returns a pointer to the array of pixels contained in a `PPix`. `char* ppixGetPixels (PPixHand pix);`

`ppixGetBounds`	Returns the bounds of a `PPix`.
	
	`void ppixGetBounds (`
	`  PPixHand pix;`
	`  prRect *bounds);`

Function	Description
ppixGetRowbytes	Returns the rowbytes of a PPix so you can properly parse the pixels returned by ppixGetPixels. int ppixGetRowbytes (PPixHand pix);
ppixNew	Allocates and returns a handle to a new PPix, with specified bounds. Since this is an older call, the pixel format is hardcoded to BGRA_4444_8u. PPixHandle ppixNew (prRect *bounds);
ppixDispose	Frees a PPixHand. void ppixDispose (PPixHand pix);
ppixLockPixels ppixUnlockPixels	These legacy functions are deprecated and should no longer be used.
ppixGetPixelAspectRatio	Passes back the pixel aspect ratio of a PPixHand. Premiere populates the longs with the PAR numerator and denominator. int ppixGetPixelAspectRatio (PPixHand pix, csSDK_uint32 *num, csSDK_uint32 *den);
ppixGetAlphaBounds	Passes back the alpha bounds of a PPixHand. void ppixGetAlphaBounds (PPixHand pix, prRect *alphaBounds);

Utility Functions

Function	Description

`getSettings`	Superceded by the Application Settings Suite. Queries Premiere for a setting, and returns its value. CS4 no longer supports many previously supported selectors.
	
	`long getSettings (long settingsSelector);`
	
	Settings values:
	`kSettingsCapDrive`
	`kSettingsTempVideo`
	`kSettingsTempAudio`
	`kSettingsProjectDrive`
	`kSettingsAudioCapDrive`
	`kSettingsProjectPath`

`getSerialNumber`	Passes back Premiere's serial number. `void getSerialNumber (char* buffer);` `buffer` - must be at least 40 characters long.

`getFileTimebase`	Passes back a file's timebase in a TDB_TimeRecord (allocated by the plug-in). If the file is already in the sequence, it is preferable to get a file's timebase using the Video Segment Suite to get the `kVideoSegmentProperty_Media_StreamFrameRate`.
	
	Note: Know your formats. Don't ask an audio only format for video, you may get unexpected results.
	
	`csSDK_int32 getFileTimebase (`
	`prFileSpec *filespec,`
	`csSDK_int32 audioOnly,`
	`TDB_TimeRecord *result);`
	
	`filespec` - description of the file, use before `getFileVideo`
	`audioOnly` - if non-zero, return the audio timebase. If zero, return the video timebase.
	`result` - the returned timebase

Function	Description

getFileVideo	Gets a frame of video (at a specified time) from a file. If the file is already in the sequence, it is preferable to get a file's video using the Clip Render Suite.
	
	`csSDK_int32 getFileVideo (`
	`  prFileSpec  *filespec,`
	`  csSDK_int32  frame,`
	`  PPixHand  thePort,`
	`  prRect  *bounds,`
	`  csSDK_int32  flags);`
	
	`filespec` - the description of the file
	`frame` - the frame to retrieve
	`thePort` - where the frame will be delivered, allocate prior to calling
	`bounds` - the boundary of the port
	`flags` - unused

getFileVideoBounds	Passes back the bounds of a file. If the file is already in the sequence, it is preferable to get a file's video bounds using the Clip Render Suite.
	
	`csSDK_int32 getFileVideoBounds (`
	`  prFileSpec  *filespec,`
	`  prRect  *bounds);`

getSPBasicSuite	This very important call returns the SweetPea suite that allows plug-ins to acquire and release all other SweetPea suites. `SPBasicSuite* getSPBasicSuite();`

getFileExtString	Passes back the list of valid entensions/filter strings given a class of media (see file types constants below).
	
	`csSDK_int32 (*plugGetFileExtStringFunc)(`
	`  csSDK_uint32 fileTypes,`
	`  char *inBuffer,`
	`  csSDK_uint32 inBufferSize);`
	
	`kFileTypes_Still` - still media
	`kFileTypes_AudioOnly` - audio-only media
	`kFileTypes_AudioVideo` - audio and video media
	`kFileTypes_AllNoIntrinsics` - all importable media types via importer plug-ins (no prproj, txt, etc)

Timeline Functions

Function	Description

`getClipVideo`	Superceded by the Clip Render Suite, which provides asynchronous import.
	
	Retrieves a frame from a clip in a segment tree returned from the Video Segment Suite. It can be used by to retrieve and store a still frame, such as a title, for playback. This call is expensive; use it carefully.
	
	`csSDK_int32 getClipVideo (`
	`  csSDK_int32  frame,`
	`  PPixHand  thePort,`
	`  prRect  *bounds,`
	`  csSDK_int32  flags,`
	`  PrClipID  clipData);`
	
	`frame` - the frame number you're requesting
	`thePort` - allocate using the PPix Creator Suite before calling
	`bounds` - the boundaries of video to return
	`flags` - either `kGCVFlag_UseFilePixelAspectRatio` or 0. Setting it to `kGCVFlag_UseFilePixelAspectRatio` will return a `PPix` stamped with the PAR of the file. Setting it to 0 will return a `PPix` adjusted to the PAR of the project and stamped accordingly. It scales, but does not stretch the `PPix` to fit the destination `PPix` that is passed in. So if the destination `PPix` is larger than the frame asked for, the frame will maintain its frame aspect ratio, letterboxing or pillarboxing the frame with transparent black. To import a frame at its native dimensions, use `getClipVideoBounds`, allocate the destination PPix using the dimensions returned, and pass the PPixHand and the dimensions into `getClipVideo`. If the frame size is not the same as the sequence size, the frame must be positioned in the composite by the plug-in.
	`clipData` - the `clipData` handle found in `prtFileRec`

Function	Description

`getWorkArea`	Passes back two `longs` with the start and end of the current work area (read-only). Set `timelineData` to the timelineData of the current sequence.
	
	`csSDK_int32 getWorkArea (`
	`PrTimelineID timelineData,`
	`csSDK_int32 *workAreaStart,`
	`csSDK_int32 *workAreaEnd);`

`getCurrentTimebase`	Passes back the current timebase of the timeline (`scale` + `sampleSize`).
	
	`void getCurrentTimebase(`
	`PrTimelineID timelineData,`
	`csSDK_uint32 *scale,`
	`csSDK_int32 *sampleSize);`
	
	`timelineData` - the `timelineData` of the current sequence
	`scale` - the sequence `scale`
	`sampleSize` - the sequence `sampleSize`

`getCurrentPos`	Returns the position of the current time indicator (the position bar set by the user). If (-1) is returned, the position bar in the timeline is not present.
	
	`csSDK_int32 getCurrentPos(`
	`PrTimelineID timelineData);`
	
	`timelineData` – the `timelineData` of the current sequence

Function	Description

getPreviewFrameEx	Gets a fully rendered frame from the timeline (all layers). Used by video filters and transitions for previews in a modal setup dialog. If the return value is -1, an error occurred, but if it is 0, the callback has returned safely. Exporters rendering final movies should NOT use this callback.
	
	`csSDK_int32 getPreviewFrameEx(`
	`PrTimelineID timelineData,`
	`csSDK_int32 inFrame,`
	`PPixHand * outRenderedFrame,`
	`const prRect * inFrameRect,`
	`PrPixelFormat * inRequestedPixelFormatArray`
	`csSDK_int32 inRequestedPixelFormatArrayCount,`
	`csSDK_uint32 inPixelAspectRatioNumerator,`
	`csSDK_uint32 inPixelAspectRatioDenominator,`
	`bool inAlwaysRender);`
	
	`timelineData` - The `timelineData` of the current sequence. Pass a timeline handle as provided in `EffectRecord`, `VideoRecord`, `compDoCompileInfo`, or `imGetPrefsRec`.
	`inFrame` - The frame to get, specified in the current timebase. If a `timelineData` handle is specified (first param above), this frame will be relative to the start of the sequence.
	`outRenderedFrame` - The destination buffer. Allocate prior to this call by the plug-in using the PPix Suite. Released by the caller before returning.

getClipVideoBounds	Passes back the dimensions of a clip in a sequence. For rolling/crawling titles, use the Roll/Crawl Suite to get the dimensions instead.
	
	`csSDK_int32 getClipVideoBounds (`
	`PrClipID inClipData,`
	`prRect *outBounds,`
	`csSDK_uint32 *outPixelAspectRatioNumerator,`
	`csSDK_uint32 *outPixelAspectRatioDenominator);`

Function	Description

getClipVideoEx	Superceded by the Clip Render Suite, which provides asynchronous import.
	
	Retrieves a frame from a clip in a segment tree returned from the Video Segment Suite. It can be used by to retrieve and store a still frame, such as a title, for playback. This call is expensive; use it carefully.
	
	`csSDK_int32 getClipVideoEx (`
	`csSDK_int32 inFrame,`
	`PPixHand *outRenderedFrame,`
	`const prRect *inFrameRect,`
	`const PrPixelFormat *inRequestedPixelFormatArray,`
	`csSDK_int32 inRequestedPixelFormatArrayCount,`
	`csSDK_uint32 inPixelAspectRatioNumerator,`
	`csSDK_uint32 inPixelAspectRatioDenominator,`
	`PrClipID inClipData);`
	
	`inFrame` - the frame number you're requesting, in the timebase of the clip
	`outRenderedFrame` - Allocated by the host. The plug-in should dispose of the `PPixHand` when done
	`inFrameRect` - the boundaries of video to return. To import a frame at its native dimensions, use `getClipVideoBounds`. If the frame size is not the same as the sequence size, the frame must be positioned in the composite by the plug-in.
	`inClipData` - the `PrClipID` from the video segment

Bottleneck Functions

The pointer to the legacy bottleneck functions is passed only to transitions and video filters. These functions are not exposed for other plug-in types. These functions are not aware of different pixel formats, and are intended only for 8-bit BGRA processing.

Sample usage:

```
((*theData)->bottleNecks->StretchBits) (*srcpix,
                                        *dstpix,
                                        &srcbox,
                                        &srcbox,
```

```
0,
NULL);
```

Function	Description

StretchBits	Stretches and copies an image, including the alpha channel. When the destination is larger than the source, it performs bilinear interpolation for smooth scaling.
	
	`void StretchBits (`
	`PPixHand srcPix,`
	`PPixHand dstPix,`
	`prRect srcRect,`
	`prRect dstRect,`
	`int mode,`
	`prRgn rgn);`
	
	`StretchBits` only works on 8-bit `PPix`s. `srcRect` is the area of the source `PPix` to copy; `dstRect` is used to scale the copy. Valid `mode`s are `cbBlend`, `cbInterp`, and `cbMaskHdl`
	
	For `cbBlend`, the low byte of the mode defines the amount of blend between the source and destination in a range of 0-255.
	
	Example:
	To blend 30% of the source with the destination, use `cbBlend | (30*255/100)`.
	
	While much slower than `cbBlend`, `cbInterp` mode does bilinear interpolation when resizing a source `PPix` to a larger destination, resulting in a much smoother image.
	
	`cbMaskHdl` tells `StretchBits` that `prRgn` is a handle to a 1-bit deep buffer the same size as the source and destination `PPix`s, to be used as a mask. Pass 0 for no clipping. The `prRgn` parameter is only used on Windows.

Function	Description

DistortPolygon	Maps the source rectangle to a four-point polygon in the destination.
	
	`void DistortPolygon (`
	`PPixHand src,`
	`PPixHand dest,`
	`prRect *srcbox,`
	`prPoint *dstpts);`
	
	When scaling up, `DistortPolygon` uses bilinear interpolation; it uses pixel averaging when scaling down.

MapPolygon	Maps a four-point src polygon into a four-point polygon (`dstpts`). If the source polygon is a rectangle, it is equivalent to `DistortPolygon`.
	
	`void MapPolygon (`
	`PPixHand src,`
	`PPixHand dest,`
	`prPoint *srcpts,`
	`prPoint *dstpts );`

DistortFixed	Equivalent to `DistortPolygon`, using fixed-point coordinates.
	
	`void DistortFixed (`
	`PPixHand src,`
	`PPixHand dest,`
	`prRect *srcbox,`
	`LongPoint *dstpts);`

FixedToFixed	Equivalent to `MapPolygon`, using fixed-point coordinates.
	
	`void FixedToFixed (`
	`PPixHand src,`
	`PPixHand dest,`
	`LongPoint *srcpts,`
	`LongPoint *dstpts);`

Function	Description

DoIndexMap	Image map function.
	
	`void DoIndexMap (`
	`  char  *src,`
	`  char  *dst,`
	`  short  row,`
	`  short,  pixwidth,`
	`  short,  height,`
	`  char  *lookup1,`
	`  char  *lookup2,`
	`  char  *lookup3);`

DoConvolve	Convolution function.
	
	`void DoConvolve (`
	`  unsigned char  *src,`
	`  unsigned char  *dst,`
	`  short  *inmatrix,`
	`  short,  rowBytes,`
	`  short,  width,`
	`  short,  height);`

4 Hardware Integration

To integrate hardware with Premiere Pro, you may consider developing up to five types of plug-ins: importers, recorders, exporters, players, and device controllers. Premiere Pro provides most of the user interface for the capture, timeline, export, and monitor windows; the plug-ins provide the functionality behind the interface.

Hardware Integration Components

Importers

Importers are used whenever frames of video or audio from a clip are needed. To give Premiere Pro the ability to read media that uses a new format or codec, develop an importer. See the Importers chapter for more information.

Recorders

Users may choose a recorder in Project > Project Settings > General > Capture Format. Recorders are used to grab frames from a hardware source and write them to a file, to be imported for editing. See the Recorders chapter for more information.

Exporters

Exporters are used whenever Premiere Pro renders preview files, or performs an export on a clip or sequence. To give Premiere Pro the ability to write media that uses a new format or codec, develop an exporter. The exporter used to render preview files in the timeline is set in Sequence > Sequence Settings > Preview File Format. The exporter used for exports is chosen when the user selects File > Export > Media > File Type. See the Exporters chapter for more information.

Players

A player drives the rendering and display of video in the Program and Source Monitors, as well as any external A/V output. To give Premiere Pro the ability to play video out to hardware for preview and final playout, write a player. The player is chosen for a sequence when the sequence is created, by the Editing Mode chosen. The player is chosen for the Source Monitor in Edit > Preferences > Player Settings. See the Players chapter for more information.

Editing Modes

An editing mode consists of a player and one or more exporters. The current editing mode is chosen by the user when creating a new sequence. The user may select a sequence preset that is associated with an editing mode (if sequence presets have been installed), or they may choose the editing mode specifically in New Sequence > General > Editing Mode. For example, when a user starts a new project, they are immediately asked to create a sequence, and presented with the New Sequence dialog. Selecting a DV preset will automatically choose the DV Editing Mode. The user can alternatively go to the General panel of the same dialog, and select the Desktop Editing Mode. This Desktop editing mode uses a different combination of a player and exporters than the DV Editing Mode.

An editing mode is defined by an XML file that references the GUID of a player and the class IDs of exporters. The XML file goes in the Editing Modes folders, which are subdirectories of language subdirectories of the Plug-ins folder. So for example:
C:\Program Files\Adobe\Adobe Premiere Pro CS5\Plug-ins\en_US\Editing Modes\SDK Editing Mode.xml

The big change with editing mode definitions in CS4 is that sequence encoder presets describe the preview file formats, rather than looking at the <EditingMode.PreviewFileFormat> tags in the editing mode XML.

See Examples\Editing Modes\SDK Editing Mode.xml for a sample XML file (not updated for CS4, but fully functional), and modify it to create a new editing mode. The description of each element is below.

```
<?xml version="1.0" encoding="utf-8" ?>
<PremiereData Version="3">
<EditingModes Version="1">
<EditingMode1 Version="1">
     <EditingMode.ID> // Create a new GUID
     <EditingMode.Name> // The localized name of your editing
          mode
     <EditingMode.Player> // The player GUID, specified by the
          player in pmStartupRec->outPlayerID.mGUID
```

```
     <EditingMode.Recorder> // The recorder GUID, specified by the
          recorder in recInfoRec->outRecorderID.mGUID
     <EditingMode.PreviewFileFormat1> // The GUID of the file for-
          mat defined later in the XML file
     <EditingMode.FrameRect1 Version="1"> // The dimensions of
          the frame size supported in the editing mode.  (0,0) if
          unrestricted
     <EditingMode.FrameRate.Count> // The number of different
          frame rates supported in the editing mode
     <EditingMode.FrameRate1 Version="1"> // A supported frame
          rate, expressed as a rational value
     <EditingMode.PAR1 Version="1"> // A supported pixel aspect
          ratio, expressed as a rational value
     <EditingMode.PAR.Count> // The number of different pixel as-
          pect ratios supported in the editing mode
</EditingMode1>
... and so on for each editing mode

<FileFormat1 Version="1">
     <FileFormat.ID> // Create a new GUID, referenced above by an
          <EditingMode.PreviewFileFormat1>
     <FileFormat.Name> // The localized name of the file format
     <FileFormat.ClassID> // The class ID, specified by the com-
          piler in compInfoRec->classID
     <FileFormat.Filetype> // The file type, specified by the com-
          piler in compInfoRec->filetype
</FileFormat1>
... and so on for each file type

<EditingModes.FileFormatCount> // The number of different file
     types supported in the editing mode
</EditingModes>

<LegacyEditingModeConverters Version="1">
<Converter1 Version="1">
     <Converter.ClassID> // The classID of the legacy editing
          mode
     <Converter.Filetype> // The file type of the legacy editing
          mode
     <Converter.Subtype> // The subtype of the legacy editing
          mode
     <Converter.ID> // The GUID of the new editing mode created
          above
     <Converter.FileFormatID> // The GUID of the new file format
          created above
```

```
     <Converter.RecorderID> // The GUID of the new recorder spec-
          ified above
</Converter1>
... and so on for each legacy editing mode converter

<LegacyEditingModeConverters.Count> // The number of different
     legacy editing modes
</LegacyEditingModeConverters>
</PremiereData>
```

ClassID, Filetype and Subtype

All plug-in types that support media must identify unique classID, filetype, and subtype. These are all four character codes, or 'fourCCs'.

Identifier	Purpose
`filetype`	Identifies the plug-in's associated file type(s). Plug-ins create lists of filetypes they support.
`subtype`	Differentiates between files of the same filetype. Identifies the codec or compression to be used.
`classID`	With the new editing mode system starting in CS4, the classID is far less important. It is used as part of the identification for exporters in the Editing Mode XML. And plug-ins may share information with most other plug-ins running in the same process using the ClassData Functions below.

ClassData Functions

All plug-in types that support media can use these callbacks to share information associated with their `classID`.

For example, these plug-ins can confirm their hardware is present and operational using the ClassData functions. They all call `getClassData` during initialization. If `getClassData` returns 0, the module checks for and initialize the hardware. It then calls `setClassData` to store information about the current context. Use handles, not pointers, for storing info.

```
typedef struct {
     SetClassDataFunc setClassData;
     GetClassDataFunc getClassData;
} ClassDataFuncs, *ClassDataFuncsPtr;
```

Function	Description

`setClassData`	Writes class data, destroys previous data.
	
	`int setClassData (`
	`  unsigned int theClass`
	`  void *info);`
	
	`theClass` - the class being set. Use a unique 4-byte code.
	`info` - the class data to be set. It can be used as a pointer or a handle. Note that all plug-ins that share the data must use the same data structure.

`getClassData`	Retrieves the class data for the given class. `int getClassData (unsigned int theClass);` `theClass` - the class for which to retrieve data.

5 Importers

Importers provide video and/or audio from the media source. This source can be a single file, a set of files, a communication link between another application, etc.

Standard importers appear as choices in the File > Import dialog, in the Files of type drop-down menu. Importers can support movies, still images, series of still images, and/or audio. If your importer provides enhanced support for a format already supported by another importer that ships with Premiere, set a high value in `imImportInfoRec.priority` to give your importer the first opportunity to handle the file.

Synthetic importers synthesize source material, rather than reading from disk. They appear in the File > New menu.

Custom importers are a special type of synthetic importer, implemented to better support titlers. Custom importers can create files on disk; synthetic importers don't. Custom importers either create new media or import existing media handled by the importer. After the file is created, the media is treated like a standard file by the host application. Additionally, the media can be modified by the importer when the user double-clicks on it in the Project Panel.

Importer Type	Reads from disk	Creates clips	Menu Location
Standard	Yes	No	File > Import
Synthetic	No	Yes	File > New
Custom	Yes	Yes	File > New File > Import

For each clip, importers can tell Premiere the resolutions and pixel formats they can decode video frames to. Premiere will request video frames as needed during scrubbing, playback, or export. Audio will be requested right when the clip is imported, if audio conforming or peak file generation is necessary. If audio conforming is not necessary, audio frames will be requested as needed during scrubbing, playback, or export. Premiere requests audio in arrays of 32-bit float, uninterleaved format.

If you've never developed an importer before, you can skip the What's New sections, and go directly to Getting Started.

What's New

What's New in Premiere Pro CS5?

When an importer's settings dialog is opened, the importer now has access to the resolution, pixel aspect ratio, timebase, and audio sample rate of the source clip, in `imGetPrefsRec`.

Custom importers can now use a new call in the Importer File Manager Suite, `RefreshFileAsync`(), to be able to update a clip after it is modified in *imGetPrefs8*.

Two new selectors have been added. *imQueryDestinationPath* allows importers that trim or copy files to be able to change the destination path of the trimmed or copy file. *imQueryContentState* gives the host an alternate way of checking the state of a clip, for clips that have multiple source files. A new return value, `inFileNotAvailable` can be returned from *imQueryContentState* if the clip is no longer available because it is offline or has been deleted.

As a convenience, when a file is opened, an importer can tell Premiere Pro how much memory to reserve for the importer's usage, rather than calling `ReserveMemory` in the Memory Manager Suite. The importer should pass back this value in `imFileOpenRec8.outExtraMemoryUsage`.

Several new return values are available for more descriptive error reporting: `imBadHeader`, `imUnsupportedCompression`, `imFileOpenFailed`, `imFileHasNoImportableStreams`, `imFileReadFailed`, `imUnsupportedAudioFormat`, `imUnsupportedVideoBitDepth`, `imDecompressionError`, and `imInvalidPreferences`.

What's New in Premiere Pro CS4?

For CS4 only, importers are loaded and called from a separate process. As a result of being in a separate process, (1) all importers must do their own file handling, (2) `privateData` is no longer accessible from *imGetPrefs8*, and (3) the compressed frame selectors such as *imGetCompressedFrame* are no longer supported (this may now be achieved using custom pixel formats and a renderer plug-in).

To debug importers, attach to the ImporterProcessServer process. There is also a separate Importer Process Plugin Loading.log.

All legacy selectors have been removed, and are now longer supported. All structures used only in these legacy selectors have been removed as well.

There are built-in XMP metadata handlers for known filetypes. These handlers write and read metadata to and from the file, without going through the importer. *imSetTimeInfo8* is no longer called, since this is set by the XMP handler for that filetype.

All file-based importers (which does not include synthetics) are required to do their own file handling now, rather than having Premiere Pro open the files. The `imCallbackFuncs: OpenFileFunc` and `ReleaseFileFunc` are no longer supported.

Due to the out-of-process importing, `privateData` is not accessible during *imGetPrefs8*, and has been removed from `imGetPrefsRec`.

imGetFrameInfo, *imDisposeFrameInfo*, *imGetCompressedFrame*, and *imDisposeCompressedFrame* are no longer supported. Supporting a custom pixel format in an importer, a renderer, and an exporter is the new way to implement smart rendering, by passing custom compressed data from input to output.

New `imFrameNotFound` return code. Returned if an importer could not find the requested frame (typically used with async importers).

New in Premiere Pro 4.1, importer prefs are now part of `imSourceVideoRec`, passed to both *imGetSourceVideo* and the async import calls

New in Premiere Pro 4.1, there is a new `filepath` member in `imFileInfoRec8`. For clips that have audio in files separate from the video file, set the filename here, so that UMIDs can properly be generated for AAFs.

What's New in Premiere Pro CS3?

Importers can specify an initial poster frame for a clip in `imImageInfoRec`.

Importers can specify subtype names during the new *imGetSubTypeNames* selector. This selector is sent after each *imGetIndFormat*, which gives an importer the opportunity to enumerate all the fourCCs and display names (e.g. "Cinepak") of their known compression types for a specific filetype. The importer can return `imUnsupported`, or create an array of `imSubTypeDescriptionRec` records (pairs of fourCCs and codec name strings) for all the codecs/subtypes it knows about.

Importers that open their own files should specify how many files they keep open between *imOpenFile8* and *imQuietFile* using the new Importer File Manager Suite, if the number is not equal to one. Importers that don't open their own files, or importers that only open a single file should not use this suite. Premiere's File Manager now keeps track of the number of files held open by

importers, and limits the number open at a time by closing the least recently used files when too many are open. On Windows, this helps memory usage, but on Mac OS this addresses a whole class of bugs that may occur when too many files are open.

Importers can also specify that certain files have very high memory usage, by setting imFileInfoRec8.highMemUsage. The number of files allowed to be open with this flag set to true is currently capped at 5.

Importers can now specify an arbitrary matte color for premultiplied alpha channels in imImageInfoRec.matteColor. Importers can state that they are uncertain about a clip's pixel aspect ratio, field type, or alpha info in imImageInfoRec.interpretationUncertain.

The imInvalidHandleValue is now -1 for Mac OS.

Importers can specify a transform matrix for frames by setting imImageInfoRec.canTransform = kPrTrue, and then during *imImportImage*, when imImportImageRec.applyTransform is non-zero, use imImportImageRec.transform, and destClipRect to calculate the transform - This code path is currently not called by Premiere Pro. After Effects uses this call to import Flash video.

New in Premiere Pro 3.1, the new capability flag, imImportInfoRec.canSupplyMetaDataClipName, allows an importer to set the clip name from metadata, rather than the file-name. The clip name should be set in imFileInfoRec8.streamName. This is useful for clips recorded by some new file-based cameras.

New in Premiere Pro 3.1, the new *imGetFileAttributes* selector allows an importer to provide the clip creation date in the new imFileAttributesRec.

What's New in Premiere Pro 2.0?

Importers can set imFileInfoRec.imImageInfoRec.alphaType to the new alphaOpaque for video with alpha channel prefilled to opaque. This allows Premiere to avoid the fill to opaque performed with video set to alphaNone.

The new Deferred Processing Suite allows an importer to schedule processing time when importing asynchronously, and to notify the user that the media item is pending additional processing.

Asynchronous Import

Video frames can now be retrieved asynchronously. The importer can optionally implement a new set of selectors.

Timecode Rate

Timecode rate can now be embedded in files. This is done for files that have timecode, but not an implicit frame rate, or where the sampling rate might differ from the timecode rate. For example, audio captured from a tape uses the video's frame rate for timecode, although its sampling rate is not equal to the timecode rate. Another example is capturing a still from tape, which could be stamped with timecode, yet not have a media frame rate. New selectors *imGetTimeInfo8* and *imSetTimeInfo8* use a new `imTimeInfoRec8` struct that has the original and alternate timecode rates. For AVI files, the timecode LIST chunk now has four new sub-chunks for two scale/sampleSize pairs.

New Selectors

imCopyFile - New selector sent rather than *imSaveFile* to importers that have set `imImportInfoRec.canCopy` when doing a copy operation using the Project Manager. The importer should maintain data on the original file rather than the copy when it returns from the selector.

imCreateAsyncImporter - New selector tells the importer to create an asynchronous importer object using the data provided, and store it in `imAsyncImporterCreationRec`.

imDeferredProcessing - New selector called to get the current progress of the deferred processing on the clip.

imGetInfo8 - New selector for unicode support, supercedes *imGetInfo7*, passing `imFileAccessRec8` instead of `imFileAccessRec` and `imFileInfo8` instead of `imFileInfo7`.

imGetPeakAudio - New selector called to get the peak values of the audio at the specified position.

imGetPreferredFrameSize - New selector called to get the frame sizes preferred by the importer.

imGetPrefs8 - New selector for unicode support, supercedes *imGetPrefs*, passing `imFileAccessRec8` instead of `imFileAccessRec`.

imGetSourceVideo - New selector called to request an unscaled frame of video. This selector will be sent instead of *imImportImage* if `supportsGetSourceVideo` is set to true during *imGetInfo8*.

imGetSupports8 - New selector called to determine if the importer supports the new Premiere Pro 2.0 selectors.

imGetTimeInfo8 - New selector called to get the file timecode and timecode rate from the file. Supercedes *imGetTimeInfo*, passing `imTimeInfoRec8` instead of `imTimeInfoRec`.

imOpenFile8 - New selector for unicode support, supercedes *imOpenFile*, passing `imOpenFileRec8` instead of `imOpenFileRec`.

imRetargetAccelerator - New selector. When the Project Manager copies media and its accelerator, this selector gives an opportunity to update the acclerator to refer to the copied media.

imSetTimeInfo8 - New selector called to set the file timecode and timecode rate to the file. Supercedes *imSetTimeInfo*, passing `imTimeInfoRec8` instead of `imTimeInfoRec`.

New/Updated Structures

imAcceleratorRec - New structure sent with the new *imRetargetAccelerator* selector.

imAsyncImporterCreationRec - New structure sent with the new *imCreateAsyncImporter* selector.

imCopyFileRec - New structure passed with new *imCopyFile* selector.

imDeferredProcessingRec - New structure sent with the new *imDeferredProcessing* selector.

imFileAccessRec8 - New structure for unicode support, supercedes `imFileAccessRec`, using a `const wchar_t[]` instead of `imFileSpec` for the file path and name. New `PrMemoryPtr newfilename` member allows synthetic importers to specify a new unicode filename during *imGetPrefs8*.

imFileInfoRec8 - New structure for unicode support, supercedes `imFileInfo7`, using a `wchar_t[]` instead of `char[]` for the stream name. New `char alwaysUnquiet` member should set to non-zero to tell Premiere if the clip should always be unquieted immediately when the application regains focus.

imFileOpenRec8 - New structure for unicode support, supercedes `imFileOpenRec`, using `imFileAccessRec8` instead of `imFileAccessRec` for the file details.

`imFileOpenRec` - Added new `long inImporterID` member to end, which can be used as the ID for calls in the PPix Cache Suite.

imFrameFormat - New structure sent with the new *imGetSourceVideo* selector, contained in `imSourceVideoRec`.

imImageInfoRec - Added new `long importerID`, `long supportsAsyncIO`, `long supportsGetSourceVideo`, `long hasPulldown`, and `long pulldownCadence` members to end. `importerID` can be used as the ID for calls in the PPix Cache Suite. `supportsAsyncIO` should be set to true if the importer supports the new asynchronous import calls. `supportsGetSourceVideo` should be set to true if the importer supports the new `imGetSourceVideo` selector. `hasPulldown` should be set to true if the clip contains NTSC film footage with 3:2 pulldown. `pulldownCadence` should be set to the enumerated value that describes the pulldown of the clip.

imImportImageRec - The `pixformat` member was changed from a `long` to a `PrPixelFormat`.

imImportInfoRec - Added new `int avoidAudioConform`, `wchar_t *acceleratorFileExt`, and `int canCopy` members to end. `avoidAudioConform` should be set to true if the importer supports fast audio retrieval and does not need the audio clips it imports to be conformed. `acceleratorFileExt` should be filled out with the file extensions of accelerator files that the importer creates and uses. `canCopy` should be set to true if the importer supports copying clips in the Project Manager.

imPeakAudioRec - New structure sent with the new *imGetPeakAudio* selector.

imPreferredFrameSizeRec - New structure sent with the new *imGetPreferredFrameSize* selector.

imIndPixelFormatRec - The `outPixelFormat` member was changed from a `long` to a `PrPixelFormat`.

imSourceVideoRec - New structure sent with the new *imGetSourceVideo* selector.

imTimeInfoRec8 - New structure holds the file timecode and timecode rate of the file. Supercedes `imTimeInfoRec`, adding scale and sampleSize members to hold the timecode rate.

imTrimFileRec8 - New structure for unicode support, supercedes `imTrimFileRec`, using a const `wchar_t[]` instead of `imFileSpec *` for the destination file path and name.

Getting Started

Try the Sample Importer Plug-ins

Choose which one of the three sample importers matches closest with your desired functionality. Build that one, or if you are still not sure, build all three! Step through the code in your debugger to learn order of events. Start your importer by modifying one of the SDK samples.

How to Get First Crack at a File

To get the first opportunity to import a filetype also supported by a built-in importer (e.g. MPEG, AVI, QuickTime, etc), provide a different `subtype and classID` in order for your importer to be called for the types of files you support. `imImportInfoRec.priority` must be higher than any of the other importers for that filetype. Set this value to 100 or higher to override all built-in importers. Premiere Pro has more than one type of AVI importer and MPEG importer,

which use this same prioritization mechanism. So your importer can override all of them and get the first shot at a filetype.

Just because you want to take over handling some files of a given filetype, it doesn't mean you have to handle all of them. To defer an unsupported `subtype` to a lower priority importer, return `imBadFile` during *imOpenFile8* or *imGetInfo8*. See the Media Abstraction chapter for more information on `filetypes`, `subtypes`, and `classIDs`.

imGetSourceVideo versus imImportImage

There are two different selectors an importer can use to provide frames to the host. Why? In a nutshell, *imGetSourceVideo* is best for media that has specific resolutions. Importers that support *imGetSourceVideo* can import frames at their native resolution or specify preferred resolutions, rather than having to scale the frames to an arbitrary size. *imImportImage* is only useful for resolution-independent video, such as vector-based graphics. Choose the one that fits the media your importer will support. The SDK importer demonstrates *imGetSourceVideo* because resolution dependent video is much more common. The synthetic importer sample demonstrates *imImportImage* because it generates video on-the-fly and doesn't have a preference as to video resolution.

privateData and prefs

Don't use global variables to store data. Use `privateData` and `prefs` instead. Each clip can have its own `privateData` and `prefs`. If the importer provides a setup dialog during *imGetPrefs8*, store any setup dialog settings in `prefs`. Store any other data in `privateData`. The host application will automatically pass the correct `privateData` and `prefs` to the appropriate importer instance.

`privateData` and `prefs` are not allocated in the same way. For `privateData`, create a handle to the custom structure you wish to store (during *imGetInfo8* or *imGetPrefs8*.) and save the handle to the `privateData` member of the structure passed in. The importer is responsible for allocating and deallocating the memory for `privateData` using Premiere's memory functions. Free the allocated `privateData` during *imCloseFile or imShutdown*, as appropriate.

For `prefs`, Premiere will allocate the `prefs` based on the `prefsLength` returned from the first call to *imGetPrefs8*. Premiere will deallocate the `prefs` when it is no longer needed.

Audio Conforming and Peak File Generation

When a clip that contains audio is imported into Premiere, one or two types of files may be generated:

First, a separate .pek file is always created, which holds peak samples for quick access when Premiere needs to draw the audio waveform.

Second, the audio may be conformed into a separate .cfa file. The conformed audio is in an interleaved 32-bit floating point format that matches the sequence audio sample rate, to maximize the speed at which Premiere can render audio effects and mix it without sacrificing quality.

Both of these files are generated through sequential calls for audio using *imImportAudio7*. Audio conforming cannot be disabled through the Premiere menus or API. However, if an importer can provide uncompressed audio of the clip at the audio sample rate of the sequence, or at an easily-converted ratio (1:2, 2:3), Premiere will not conform the audio. All compressed audio data must be conformed.

The location of the .cfa and .pek files is determined by the user-specified path in Edit > Preferences > Media > Media Cache Files. When the project is closed, the files will be cleaned up. If the source clip is not saved in the project, the associated conformed audio files will be deleted.

Importers can get audio for scrubbing, playing and other timeline operations before conforming has completed, resulting in responsive audio feedback during conforming. To do this, they must support both random access and sequential access audio importing. The `kSeparateSequentialAudio` access mode should be set in `imFileInfoRec8.accessModes`.

Quieting versus Closing a File

When the application loses focus, importers receive *imQuietFile* for each file it has been asked to open. Update any `PrivateData` and close the file. If the project is closed, *imCloseFile* is sent, telling the importer to free any `PrivateData`. If the importer didn't store any `PrivateData`, it will not receive *imCloseFile*.

File Handling

Importers can provide their own file handling (useful if you have child files or a custom file system). Set `canOpen`, `canSave`, and `canDelete` to true during *imInit*, and respond to *imOpenFile8*, *imQuietFile*, *imCloseFile*, *imSaveFile8*, *imDeleteFile8*. Use the Async File Reader Suite for cross-platform file operations.

Quality Levels

Importers can optionally support two different quality modes, with the `imDraftMode` flag that is used in `imImportImageRec`.

Multiple Audio Streams

Importers can support multiple streams of audio and/or video. This is useful for audio configurations beyond mono, stereo, and 5.1, or for stills with layers, such as Photoshop files. An importer describes each stream one-by-one during iterative calls to *imGetInfo8*. In response to each call, the importer describes one stream, and returns `imIterateStreams`, until it reaches the last stream, and then it returns `imBadStreamIndex`. Set `imFileInfoRec8->streamsAsComp = kPrFalse`, so that the set of streams appear as a single clip within Premiere Pro.

In *imGetInfo8*, save `streamIdx` in `privateData`, to have access to it later. That way, when called in *imImportAudio7*, the importer will know which stream of audio to pass back.

See the sample code in the SDK File Importer, which can be turned on by uncommenting back in the `MULTISTREAM_AUDIO_TESTING` define in SDK_File_Import.h.

Project Manager Support

The Project Manager is only in Premiere Pro, and it allows users to archive projects, trim out unused media, or collect all source files to a single location. Importers are the most knowledgable about the sources they work with. So Premiere Pro doesn't make any assumptions about the source media, but instead relies on the importers to handle the trimming and file size estimates. Only importers that specifically support trimming will trim and not copy when the Project Manager trims projects.

To support trimming, importers will want to set the `canCalcSizes` and `canTrim` flags during *imInit*, and support *imCalcSize8*, *imCheckTrim8*, and *imTrimFile8*.

If the each clip has more than one source file (such as audio channels in separate files), the importer should also set `canCopy` and support *imCopyFile*. Otherwise, the Project Manager will not know about the other source files.

External files, such as textures, logos, etc. that are used by an importer instance but do not appear as footage in Project panel, should be registered with Premiere Pro using the File Registration Suite during *imGetInfo8* or *imGetPrefs8*. Registered files will be taken into account when trimming or copying a project using the Project Manager.

Creating a Custom Importer

This variant of the importer API allows importers to dynamically create disk-based media while working within Premiere. A titler plug-in or similar should use this API. Once your clip is created, it is treated like any other standard file and will receive all standard missing file handling.

A Custom Importer ***must*** do the following:

- Set the following flags true in `imImportInfoRec`; `canCreate`, `canOpen`, `addToMenu`, `noFile`. This tells Premiere your plug-in will create a clip on disk at *imGetPrefs8* time.

- To determine when you need to create a new clip vs. modify an existing clip, check the `imFileAccessRec` filename. If it's identical to the plug-in display name (as set in the PiPL), create a new clip; otherwise modify the clip.

- If the user cancels from your dialog when creating a new clip, return `imCancel`.

- If the clip is modified, the importer needs to do a few things for Premiere to pick up the changes. Put your file access information in the supplied `imFileAccessRec`. Premiere will use this data to reference your clip from now on. Close the file handle after you create it. Return `imSetFile` after creating a file handle in *imGetPrefs8*., and call `RefreshFileAsync`() in the Importer File Manager Suite to notify Premiere that the clip has been modified. Premiere will immediately call you to open the file and return a valid `imFileRef`. Respond to *imOpenFile8*, *imQuietFile*, *imCloseFile* at a minimum.

Showing a Video Preview in a Setup Dialog

Synthetic importers can get a frame from the timeline beneath the current clip or timeline location. This is useful for titler plug-ins. Use the `getPreviewFrameEx` callback with the time given by `TDB_TimeRecord tdbTimelocation` in `imGetPrefsRec`. `timelineData` is now also valid during *imGetPrefs8*.

Real-Time Rolling and Crawling Titles

For RT rolls and crawls, a player and importer must be specially designed to work together. An importer can implement the appropriate functionality, but it is up to the player to take advantage of it.

Importers can make image data available for rolling and crawling titles, using `imImageInfoRec.isRollCrawl`. If the importer sets it to non-zero, this declares that the image is a title or other image that does roll/crawl, and that the importer supports the *imGetRollCrawlInfo* and *imRollCrawlRenderPage* selectors. *imGetRollCrawlInfo* is used to get info on the roll/crawl from the importer, and *imRollCrawlRenderPage* is used to get a rendered page of the roll/crawl.

Format repeated in menu?

To avoid having your importer appear multiple times in the file formats supported pop-up list, fill out the `formatName`, `formatShortName` and platform extension once and only once during your *imGetIndFormat*.

Resources

Importers must contain a IMPT resource. Premiere uses this to identify the plug-in as an importer. Also, depending on the type of importer (standard, synthetic, or custom), a PiPL may be required.

Entry Point

```
csSDK_int32 xImportEntry (
     csSDK_int32    selector,
     imStdParms     *stdParms,
     void           *param1,
     void           *param2)
```

selector is the action Premiere wants the importer to perform. `stdParms` provides callbacks to obtain additional information from Premiere or to have Premiere perform tasks. `param1` and `param2` vary with the selector; they may contain a specific value or a pointer to a structure. Return `imNoErr` if successful, or an appropriate return code.

Standard Parameters

A pointer to this structure is sent from the host application to the plug-in with every selector.

```
typedef struct {
     csSDK_int32         imInterfaceVer;
     imCallbackFuncs     *funcs;
     piSuitesPtr         piSuites;
} imStdParms;
```

Member	Description
`imInterfaceVer`	Importer API version Premiere Pro CS5 - `IMPORTMOD_VERSION_10` Premiere Pro CS4 - `IMPORTMOD_VERSION_9` Premiere Pro CS3 - `IMPORTMOD_VERSION_8`

funcs	Pointers to callbacks for importers
piSuites	Pointer to universal callback suites

Importer-Specific Callbacks

```
typedef struct {
     ClassDataFuncsPtr   classFuncs;
     csSDK_int32         unused1;
     csSDK_int32         unused2;
} imCallbackFuncs;

typedef csSDK_int32 (*importProgressFunc){
     csSDK_int32    partDone;
     csSDK_int32    totalToDo;
     void           *trimCallbackID};
```

Function	Description
classFuncs	See ClassData functions.
importProgressFunc	Available in imSaveFileRec and imTrimFileRec during *imSaveFile8* and *imTrimFile8*. Callback function pointer for use during project archiving or trimming to call into Premiere and update the progress bar and check for cancellation. Either imProgressAbort or imProgressContinue will be returned. The trimCallbackID parameter is passed in the same structures.

Selector Table

This table is ordered roughly by calling sequence. The Synth column indicates whether or not the selector is applicable to synthetic importers. Custom importers can respond to any of the selectors.

Selector	param1	param2	Synth
imInit	imImportInfoRec*	unused	Yes
imGetIndFormat	(int) index	imIndFormatRec*	Yes
imGetSubTypeNames	(csSDK_int32) fileType	imSubTypeDescriptionRec **	No
imGetIndPixelFormat	(int) index	imIndPixelFormatRec*	Yes

imGetPrefs8	imFileAccessRec8*	imGetPrefsRec*	Yes
imGetInfo8	imFileAccessRec8*	imFileInfoRec8*	Yes
imGetTimeInfo8	imFileRef	imTimeInfoRec8*	No
imSetTimeInfo8	imFileRef	imTimeInfoRec8*	No
imGetFileAttributes	imFileAttributesRec*	unused	
imImportImage	imFileRef	imImportImageRec*	Yes
imGetPreferredFrameSize	imPreferredFrame-SizeRec*	unused	Yes
imGetSourceVideo	imFileRef	imSourceVideoRec*	Yes
imImportAudio7	imFileRef	imImportAu-dioRec7*	Yes
imResetSequentialAudio	imFileRef	imImportAu-dioRec7*	Yes
imGetSequentialAudio	imFileRef	imImportAu-dioRec7*	Yes
imGetPeakAudio	imFileRef	imPeakAudioRec*	Yes
imOpenFile8	imFileRef*	imFileOpenRec8*	No
imQuietFile	imFileRef*	(void*) PrivateData**	No
imCloseFile	imFileRef*	(void*) PrivateData**	No
imSaveFile8	imSaveFileRec8*	unused	No
imAnalysis	imFileRef	imAnalysisRec*	Yes
imDataRateAnalysis	imFileRef	imDataRate-AnalysisRec*	No
imDeleteFile	imDeleteFileRec*	unused	No
imGetMetaData	imFileRef	imMetaDataRec*	No
imSetMetaData	imFileRef	imMetaDataRec*	No
imShutdown	unused	unused	Yes
imGetSupports8	unused	unused	Yes
imGetSupports7	unused	unused	Yes
imGetRollCrawlInfo	imRollCrawl-InfoRec*	unused	Yes
imRollCrawlRenderPage	rollCrawlRender-Rec*	unused	Yes
imCalcSize8	imCalcSizeRec*	imFileAccessRec8*	No
imCheckTrim8	imCheckTrimRec*	imFileAccessRec8*	No
imTrimFile8	imFileAccessRec8*	imTrimFileRec8*	No
imCopyFile	imCopyFileRec*	unused	No

imDeferredProcessing	imDeferred-ProcessingRec *	unused	No
imRetargetAccelerator	imAcceleratorRec *	unused	No
imCreateAsyncImporter	imAsyncImporter-CreationRec *	unused	Yes
imQueryDestinationPath	imQueryDestination-PathRec *	unused	No
imQueryContentState	imQueryContent-StateRec *	unused	No

Selector Descriptions

This section provides a brief overview of each selector and highlights implementation issues. Additional implementation details are at the end of the chapter.

imInit

```
param1 - imImportInfoRec *
param2 - unused
```

Sent during application startup. Describe the importer's capabilities in the imImportInfoRec; all options are false by default. The similarly named flags in imIndFormatRec.flags are obsolete and should not be used.

Set hasSetup to kPrTrue if the importer has a setup dialog, and setupOnDblClk to kPrTrue to have that dialog display when the user double-clicks a file in the Project Panel; Premiere throws away any preview files generated for a file imported with this setting, even if no setup dialog is displayed.

Return imIsCacheable from *imInit* if a plug-in does not need to be called to initialize every time Premiere ilaunched. This will help reduce the time to launch the application.

Synthetic Importers

Setting noFile to kPrTrue indicates that the importer is synthetic. Set addToMenu to kPrTrue to add the importer to the File > New menu.

Custom Importers

To create a custom importer, the following capabilities must be set. See Additional Details for more info on custom importers.

```
noFile = kPrTrue;
hasSetup = kPrTrue;
canOpen = kPrTrue;
canCreate = kPrTrue;
addToMenu = imMenuNew;
```

imGetIndFormat

```
param1 - (int) index
param2 - imIndFormatRec *
```

Sent repeatedly, immediately after imInit; enumerate the filetypes the plug-in supports by populating the imIndFormatRec. When finished, return imBadFormatIndex. imIndFormatRec.flags are obsolete and should not be used.

Synthetic Importers
Because they have no file, synthetic importers only need to respond with the filetype established in their resource. Create a separate plug-in for each synthetic file type.

imGetSubTypeNames

```
param1 - (csSDK_int32) fileType
param2 - imSubTypeDescriptionRec **
```

New optional selector added for After Effects CS3. As of CS4, this info still isn't used in Premiere Pro, but is used in After Effects to display the codec name in the Project Panel. The importer should fill in the codec name for the specific subtype fourcc provided. This selector will be sent repeatedly until names for all subtypes have been requested. The `imSubTypeDescriptionRec` must be allocated by the importer, and will be released by the plug-in host.

imGetIndPixelFormat

```
param1 - (int) index
param2 - imIndPixelFormatRec *
```

New optional selector called to enumerate the pixel formats available for a specific file. This message will be sent repeatedly until all formats have been returned. Pixel formats should be returned in the preferred order that the importer supports. The Importer should return imBadFormatIndex after all formats have been enumerated. imUnsupported should be returned on the first call if only *yawn* `BGRA_4444_8u` is supported.

What pixel formats should you support? Keep it real. Just return the pixel format that most closely matches the data stored in your file. If decoding to two or more formats can be done at about the same speed, declare support for both, but favor any pixel formats that are more compact, to save on memory and bandwidth.

imGetPrefs8

```
param1 - imFileAccessRec8 *
param2 - imGetPrefsRec *
```

New in Premiere Pro 2.0. Premiere sends this selector when the user imports (or creates, if synthetic) a file of your type. You must have set `hasSetup = true` during *imInit* to receive this selector.

Storing preferences is a two step process. If the pointer in `imGetPrefsRec.prefs` is NULL, set `prefsLength` to the size of your preferences structure and return `imNoErr`. Premiere sends *imGetPrefs* again; display your dialog, and pass the preferences pointer back in `imGetPrefsRec.prefs`. Starting in Premiere Pro 1.5, the importer can get a frame from the timeline beneath the current clip or timeline location. This is useful for titler plug-ins. Use the getPreviewFrameEx callback with the time given by TDB_TimeRecord tdbTimelocation in imGetPrefsRec.

Synthetic Importers
Synthetic importers can specify the displayable name by changing the newfilename member `imFileAccessRec8`.

The first time this selector is sent, the `imGetPrefsRec.timelineData`, though non-null, contains garbage and should not be used. It will only contain valid information once the user has put the clip into the timeline, and is double-clicking on it.

Custom Importers
Custom importers should return `imSetFile` after successfully creating a new file, storing the file access information in `imFileAccessRec8`. Premiere will use this data to then ask the importer to open the file it created. See Additional Details for more information on custom importers.

imGetInfo8

```
param1 - imFileAccessRec8 *
param2 - imFileInfoRec8 *
```

New in Premiere Pro 2.0. Replacement for *imGetInfo7* that uses new `imFileInfoRec8`. Called when a specific file is instantiated. Importer checks file validity, optionally allocates file instance

data, and describes the properties of the file being imported by populating the `imFileInfoRec8`.

Synthetic Importers
You can create a still frame, a movie of a set duration, or an 'infinite' length movie, but cannot change the properties of a synthetic file once imported.

imGetTimeInfo8

```
param1 - imFileRef
param2 - imTimeInfoRec8 *
```

New in Premiere Pro 2.0. Read any embedded timecode data in the file. Supercedes *imGetTimeInfo*.

imSetTimeInfo8

```
param1 - imFileRef
param2 - imTimeInfoRec8 *
```

New in Premiere Pro 2.0. Sent after a capture completes, where timecode was provided by the recorder or device controller. Use this to write timecode data and timecode rate to your file. See the Universals chapter for more information on time in Premiere. Supercedes *imSetTimeInfo*.

Timecode rate is important for files that have timecode, but not an implicit frame rate, or where the sampling rate might differ from the timecode rate. For example, audio captured from a tape uses the video's frame rate for timecode, although its sampling rate is not equal to the timecode rate. Another example is capturing a still from tape, which could be stamped with timecode, yet not have a media frame rate.

imGetFileAttributes

```
param1 - imFileAttributesRec *
```

New in Premiere Pro 3.1. Optional. Pass back the creation date stamp in `imFileAttributesRec`.

imImportImage

```
param1 - imFileRef
param2 - imImportImageRec *
```

Give the host a frame of video; populate the `imImportImageRec` buffer with pixel data, or (if you've set `canDraw` to true during *imInit*) draw to the screen in the provided window using platform-specific calls to do so. You must scale the image data to fit the window; Premiere relies on the import module for properly scaled frames.

imGetPreferredFrameSize

```
param1 - imFileRef
param2 - imPreferredFrameSizeRec *
```

New in Premiere Pro 2.0. Provide the frame sizes preferred by the importer.

imGetSourceVideo

```
param1 - imFileRef
param2 - imSourceVideoRec *
```

New in Premiere Pro 2.0. Get the host an unscaled frame of video. This selector will be sent instead of *imImportImage* if `supportsGetSourceVideo` is set to true during *imGetInfo8*.

imImportAudio7

```
param1 - imFileRef
param2 - imImportAudioRec7 *
```

Replacement for imImportAudio that uses new imAudioInfoRec7. Called to import audio using the new 32-bit float, uninterleaved audio format. Fill `imImportAudioRec7->buffer` with the number of sample frames specified in `imImportAudioRec7->size`, starting from `imImportAudioRec7->position`. Always return 32-bit float, uninterleaved samples as described in the Universals chapter. You may use the calls in the Audio Suite to do some common conversions.

imGetPeakAudio

```
param1 - imFileRef
param2 - imPeakAudioRec *
```

Optional selector allows Premiere to get audio peak data directly from the importer. This is used for synthetic clips longer than five minutes. Providing peak data can significantly improve waveform rendering performance when the user views audio waveform of the clip in the Source Monitor.

imOpenFile8

```
param1 - imFileRef *
param2 - imFileOpenRec8 *
```

New in Premiere Pro 2.0. Open a file and give Premiere its handle. This selector is sent only if `canOpen` was set to true during *imInit*; do so if you generate child files, you need to have read and write access during the Premiere session, or use a custom file system. If you respond to this selector, you must also respond to *imQuietFile* and *imCloseFile*. You may additionally need to respond to *imDeleteFile* and *imSaveFile*; see Additional Details. Close any child files during *imCloseFile*.

Importers that open their own files should specify how many files they keep open between *imOpenFile8* and *imQuietFile* using the new Importer File Manager Suite, if the number is not equal to one. Importers that don't open their own files, or importers that only open a single file should not use this suite. Premiere's File Manager now keeps track of the number of files held open by importers, and limits the number open at a time by closing the least recently used files when too many are open. On Windows, this helps memory usage, but on Mac OS this addresses a whole class of bugs that may occur when too many files are open.

imQuietFile

```
param1 - imFileRef *
param2 - (void*) PrivateData **
```

Close the file in `imFileRef`, and release any hardware resources associated with it. Respond to this selector only if *canOpen* was set to true during imInit. A quieted file is closed (at the OS level), but associated `privateData` is maintained by Premiere. Do not deallocate `privateData` in response to *imQuietFile*; do so during *imCloseFile*.

imCloseFile

```
param1 - imFileRef *
param2 - (void*) PrivateData **
```

The specified file is no longer required; dispose of `privateData`. Only sent if `privateData` was allocated during *imGetInfo*.

imSaveFile8

```
param1 - imSaveFileRec8 *
param2 - unused
```

Save the file specified in imSaveFileRec8. Only sent if canOpen was set to true during *imInit*.

imAnalysis

```
param1 - imFileRef
param2 - imAnalysisRec *
```

Provide information about the file in the imAnalysisRec; this is sent when the user chooses Get Properties on your file. Premiere displays a dialog with information about the file, including the text you provide.

imDataRateAnalysis

```
param1 - imFileRef
param2 - imDataRateAnalysisRec *
```

Give Premiere a data rate analysis of the file. Sent when the user presses the Data Rate button in the Get Properties dialog, and is enabled only if hasDataRate was true in the imFileInfoRec returned during *imGetInfo*. Premiere generates a data rate analysis graph from the data provided.

imDeleteFile

```
param1 - imDeleteFileRec *
param2 - unused
```

Request this selector (by setting canDelete to true during *imInit*) only if you have child files or related files associated with your file. If you have only a single file per clip you do not need to delete your own files. Numbered still file importers do not need to respond to this selector; each file is handled individually.

imGetMetaData

```
param1 - imFileRef
param2 - imMetaDataRec *
```

Called to get a metadata chunk specified by a fourcc code. If imMetaDataRec->buffer is null, the plug-in should set buffersize to the required buffer size and return imNoErr. Premiere will then call again with the appropriate buffer already allocated.

imSetMetaData

```
param1 - imFileRef
param2 - imMetaDataRec *
```

Called to add a metadata chunk specified by a fourcc code.

imShutdown

```
param1 - unused
param2 - unused
```

Release all resources and perform any other necessary clean-up; sent when Premiere quits.

imGetSupports8

```
param1 - unused
param2 - unused
```

A plug-in that supports the Premiere Pro 2.0 API must return `malSupports8`.

imGetSupports7

```
param1 - unused
param2 - unused
```

A plug-in that supports the Premiere Pro 1.0 API must return `malSupports7`.

imCalcSize8

```
param1 - imCalcSizeRec *
param2 - imFileAccessRec8 *
```

Called before Premiere trims a clip, to get the disk size used by a clip. This selector is called if the importer sets `imImportInfoRec.canCalcSizes` to non-zero. An importer should support this call if it uses a tree of files represented as one top-level file to Premiere. The importer should calculate either the trimmed or current size of the file and return it. If the trimIn and duration are set to zero, Premiere is asking for the current size of the file. If the trimIn and duration are valid values, Premiere is asking for the trimmed size.

imCheckTrim8

```
param1 - imCheckTrimRec *
param2 - imFileAccessRec8 *
```

Called before Premiere trims a clip, to check if a clip can be trimmed at the specified boundaries. imCheckTrimRec and imFileAccessRec are passed in. The importer examines the proposed trimmed size of the file, and can change the requested in point and duration to new values if the file can only be trimmed at certain locations (for example, at GOP boundaries in MPEG files). If the importer changes the in and duration, the new values must include all the material requested in the original trim request. If an importer does not need to change the in and duration, it may either return imUnsupported, or imNoErr and simply not change the in/duration fields. If the importer does not want the file trimmed (perhaps because the audio or video would be degraded if trimmed at all) it can return imCantTrim and the trim operation will fail and the file will be copied instead.

For a file with both audio and video, the selector will be sent several times. The first time, `imCheckTrimRec` will have both `keepAudio` and `keepVideo` set to a non-zero value, and the trim boundaries will represent the entire file, and Premiere is asking if the file can be trimmed at all. If the importer returns an error, it will not be called again. The second time, `imCheckTrimRec` will have `keepAudio` set to a non-zero value, and the trim boundaries will represent the audio in and out points in the audio timebase, and Premiere is asking if the audio can be trimmed on these boundaries. The third time, `imCheckTrimRec` will have `keepVideo` set to a non-zero value, and the trim boundaries will represent the video in and out points in the video timebase, and Premiere is asking if the video can be trimmed on these boundaries. If either the video or audio boundaries extend further than the other boundaries, Premiere will trim the file at the furthest boundary.

imTrimFile8

```
param1 - imFileAccessRec8 *
param2 - imTrimFileRec8 *
```

New in Premiere Pro 2.0. Called when Premiere trims a clip. `imFileAccessRec8` and `imTrimFileRec8` are passed in. `imDiskFull` or `imDiskErr` may be returned if there is an error while trimming. The current file, inPoint, and new duration are given and a destination file path. If a file has video and audio, the trim time is in the video's timebase. For audio only, the trim times are in the audio timebase. A simple callback and callbackID is passed to imTrimFile8 and imSaveFile8 that allows plug-ins to query whether or not the user has cancelled the operation. If non-zero (and they can be nil), the callback pointer should be called to check for cancellation. The callback function will return `imProgressAbort` or `imProgressContinue`.

imCopyFile

`param1` - `imCopyFileRec *`
`param2` - unused

New in Premiere Pro 2.0. *imCopyFile* is sent rather than *imSaveFile* to importers that have set `imImportInfoRec.canCopy` when doing a copy operation using the Project Manager. The importer should maintain data on the original file rather than the copy when it returns from the selector.

imDeferredProcessing

`param1` - `imDeferredProcessingRec *`
`param2` - unused

New in Premiere Pro 2.0. Describe the current progress of the deferred processing on the clip.

imRetargetAccelerator

`param1` - `imAcceleratorRec *`
`param2` - unused

New in Premiere Pro 2.0. When the Project Manager copies media and its accelerator, this selector gives an opportunity to update the acclerator to refer to the copied media.

imCreateAsyncImporter

`param1` - `imAsyncImporterCreationRec *`
`param2` - unused

New in Premiere Pro 2.0. Create an asynchronous importer object using the data provided, and store it in imAsyncImporterCreationRec.

imQueryDestinationPath

param1 - `imQueryDestinationPathRec` *
param2 - unused

New in CS5. This allows the plug-in to modify the path that will be used for a trimmed clip, so the trimmed project is written with the correct path.

imQueryContentState

param1 - imQueryContentStateRec *
param2 - unused

New in CS5. This is used by streaming importers and folder based importers (P2, XDCAM, etc) that have multiple files that comprise the content. If an importer doesn't support the selector then the host checks the last modification time for the file.

Return Codes

Return Code	Reason
imNoErr	Operation has completed without error.
imTooWide	File dimensions too large.
imBadFile	Bad file format. To defer an unsupported subtype to a lower priority importer, return this during *imOpenFile8* or *imGetInfo8*.
imUnsupported	Unsupported selector.
imMemErr	Memory error.
imOtherErr	Unknown error.
imNoContent	No audio or video.
imBadRate	Bad audio rate.
imBadCompression	Bad compression.
imBadCodec	Codec not found.
imNotFlat	Unflattened QuickTime movie.
imBadSndComp	Bad sound compression.
imNoTimecode	Timecode supported, but not found.
imMissingComponent	Missing component needed to open the file.
imSaveErr	Error saving file.
imDeleteErr	Error deleting file.
imNotFoundErr	The requested metadata chunk was not found.
imSetFile	Return this from *imGetPrefs8* only if you are a custom importer and you need Premiere to alter it's file access information (e.g. a new path or filename is created).
imIterateStreams	Return from *imGetInfo8* to indicate that there are more streams to describe. Premiere will send *imGetInfo8* for the next stream.
imBadStreamIndex	Return from *imGetInfo8* after interating through streams to indicate that there are no more streams to describe.
imCantTrim	Return from *imCheckTrim* if the file cannot be trimmed by the importer.

imDiskFull	Return from *imTrimFile8* if there is not enough space to complete the trim operation.
imDiskErr	Return from *imTrimFile8* if there is a general disk or I/O error during the trim operation.
imFileShareViolation	Return from *imOpenFile8* if file cannot be opened due to another process having exclusive read access
imIterateFrameSizes	Return from *imGetPreferredFrameSize*, to be called again to describe more frame sizes for a particular pixel format.
imMediaPending	Return from *imGetSourceVideo* or *imCreateAsyncImporter* if the importer is still processing the file and cannot return video frames yet.
imDRMControlled	Return from *imOpenFile8* if the file cannot be opened because it is under rights management.
imActivationFailed	Activation of a component failed (usually due to user cancellation). This is used by Premiere Elements.
imFrameNotFound	New in CS4. Return if an importer could not find the requested frame (typically used with async importers)
imBadHeader	New in CS5. The file cannot be opened because of a header error
imUnsupportedCompression	New in CS5. The file has a compression type that the importer does not support
imFileOpenFailed	New in CS5. The importer was unable to open the file on disk
imFileHasNoImportableStreams	New in CS5. The file has no audio or video stream
imFileReadFailed	New in CS5. A read from an open file failed
imUnsupportedAudioFormat	New in CS5. The importer cannot import something in the audio format
imUnsupportedVideoBitDepth	New in CS5. The video bit depth of this file is unsupported by the importer
imDecompressionError	New in CS5. The importer hit an error decompressing the audio or video
imInvalidPreferences	New in CS5. Invalid prefs data was passed to the importer
inFileNotAvailable	New in CS5. Return from *imQueryContentState* if the file/stream is no longer available because it is offline or deleted
imCancel	Return from *imGetPrefs8* if user cancels or the plug-in cannot open the file (custom/synthetic importer).
imBadFormatIndex	Return this when given an out of range format index, and from *imGetIndFormat* when plug-in has no more formats to enumerate.

imIsCacheable	Return from *imInit* if a plug-in does not need to be called to initialize every time Premiere is launched. This will help reduce the time to launch the application.

Structures

Structure	Sent with selector
imAcceleratorRec	*imRetargetAccelerator*
imAnalysisRec	*imAnalysis*
imAsyncImporterCreationRec	*imCreateAsyncImporter*
imAudioInfoRec7	*imGetInfo8* (member of imFileInfoRec7)
imCalcSizeRec	*imCalcSize8*
imCheckTrimRec	*imCheckTrim8*
imCopyFileRec	*imCopyFile*
imDataRateAnalysisRec	*imDataRateAnalysis*
imDeferredProcessingRec	*imDeferredProcessing*
imDeleteFileRec	*imDeleteFile*
imFileAccessRec8	*imGetInfo8* and *imGetPrefs8*
imFileAttributesRec	*imGetFileAttributes*
imFileInfoRec8	*imGetInfo8*
imFileOpenRec8	*imOpenFile8*
imFileRef	*imAnalysis*, *imDataRateAnalysis*, *imOpenFile8*, *imQuietFile*, *imCloseFile*, *imGetTimeInfo8*, *imSetTimeInfo8*, *imImportImage*, *imImportAudio7*
imFileSpec	*imGetInfo8*, *imGetPrefs8*, *imSaveFile8*, *imDeleteFile*, and *imTrimFile8* (member of imFileAccessRec8, imSaveFileRec8, imDeleteFileRec, and imTrimFileRec8)
imFrameFormat	*imGetSourceVideo* (member of imSourceVideoRec)
imGetPrefsRec	*imGetPrefs8*
imImageInfoRec	*imGetInfo8* (member of imFileInfoRec8)
imImportAudioRec7	*imImportAudio7*
imImportImageRec	*imImportImage*
imImportInfoRec	*imInit*
imIndFormatRec	*imGetIndFormat*
imIndPixelFormatRec	*imGetIndPixelFormat*
imMetaDataRec	*imGetMetaData* and *imSetMetaData*
imPeakAudioRec	*imGetPeakAudio*

imPreferredFrameSizeRec	*imGetPreferredFrameSize*
imQueryContentStateRec	
imQueryDestinationPath-Rec	
imRollCrawlInfoRec	*imGetRollCrawlInfo*
imRollCrawlRenderRec	*imRollCrawlRenderPage*
imSaveFileRec8	*imSaveFile8*
imSourceVideoRec	*imGetSourceVideo*
imSubTypeDescriptionRec	*imGetSubTypeNames*
imTimeInfoRec8	*imGetTimeInfo8* and *imSetTimeInfo8*
imTrimFileRec8	*imTrimFile8*

Structure Descriptions

imAcceleratorRec

Selector: *imRetargetAccelerator*

Describes the path to the new media and new accelerator created when the Project Manager copies media and its accelerator.

```
typedef struct {
     const prUTF16Char   *inOriginalPath;
     const prUTF16Char   *inAcceleratorPath;
} imAcceleratorRec;
```

inOriginalPath	The unicode path and name of the copied media.
inAcceleratorPath	The unicode path and name of the copied accelerator.

imAnalysisRec

Selector: *imAnalysis*

Sending back analysis data is a two step process. First, set `buffersize` to the size of your character buffer and return `imNoErr`. Premiere will immediately send *imAnalysis* again; populate the buffer with text. Previously-stored preferences and `privateData` are returned in this structure.

```
typedef struct {
```

```
     void           *privatedata;
     void           *prefs;
     csSDK_int32    buffersize;
     char           *buffer;
     csSDK_int32    *timecodeFormat;
} imAnalysisRec;
```

privatedata	Instance data from *imGetInfo8* or *imGetPrefs8*.
prefs	Instance data from *imGetPrefs8* (setup dialog info).
buffersize	Set to the desired size and return imNoErr to Premiere, which will re-size and call the plug-in again with the *imGetPrefs8* selector.
buffer	Text buffer. Terminate lines with line endings (CR and LF).
timecodeFormat	Preferred timecode format, sent by the host.

imAsyncImporterCreationRec

Selector: *imCreateAsyncImporter*

Create an asynchronous importer object using the data provided, and store it here.

```
typedef struct {
     void                *inPrivateData;
     void                *inPrefs;
     AsyncImporterEntry  outAsyncEntry;
     void                *outAsyncPrivateData;
}
```

inPrivateData	Instance data from *imGetInfo8* or *imGetPrefs8*.
inPrefs	Instance data from *imGetPrefs8* (setup dialog info).
outAsyncEntry	Provide the entry point for async selectors sent to the asynchronous importer object.
outAsyncPrivate-Data	PrivateData for the asynchronous importer object.

imAudioInfoRec7

Selector: *imGetInfo8* (member of imFileInfoRec8)

Audio data properties of the file (or of the data you will generate, if synthetic).

```
typedef struct {
     csSDK_int32         numChannels;
```

```
    float                   sampleRate;
    PrAudioSampleType       sampleType;
}
```

numChannels	Number of audio channels in the audio stream. Either 1, 2, or 6.
sampleRate	In hertz.
sampleType	This is for informational use only, to disclose the format of the audio on disk, before it is converted to 32-bit float, uninterleaved, by the importer. The audio sample types are listed in the Universals chapter.

imCalcSizeRec

Selector: *imCalcSize8*

Asks the importer for an estimate of disk space used by the clip, given the provided trim boundaries.

```
typedef struct {
    void            *privatedata;
    void            *prefs;
    csSDK_int32     trimIn;
    csSDK_int32     duration;
    prInt64         sizeInBytes;
    csSDK_int32     scale;
    csSDK_int32     sampleSize;
} imCalcSizeRec;
```

privatedata	Instance data gathered from *imGetInfo8* or *imGetPrefs8*.
prefs	Instance data gathered from *imGetPrefs8* (setup dialog info).
trimIn	In point of the trimmed clip the importer should calculate the size for, in the timebase specified by scale over sampleSize.
duration	Duration of the trimmed clip the importer should calculate the size for. If 0, then the importer should calculate the size of the untrimmed clip.
sizeInBytes	Return the calculated size in bytes.
scale	The frame rate of the video clip, represented as scale over sampleSize.
sampleSize	

imCheckTrimRec

Selector: *imCheckTrim8*

Provides the requested trim boundaries to the importer, and allows adjusted trim boundaries to be passed back to Premiere.

```
typedef struct {
     void           *privatedata;
     void           *prefs;
     csSDK_int32    trimIn;
     csSDK_int32    duration;
     csSDK_int32    keepAudio;
     csSDK_int32    keepVideo;
     csSDK_int32    newTrimIn;
     csSDK_int32    newDuration;
     csSDK_int32    scale;
     csSDK_int32    sampleSize;
} imCheckTrimRec;
```

privatedata	Instance data gathered from *imGetInfo8* or *imGetPrefs8*.
prefs	Instance data gathered from *imGetPrefs8* (setup dialog info).
trimIn	Requested in point of the trimmed clip, in the timebase specified by scale over sampleSize.
duration	Requested duration. If 0, then the request is to leave the clip untrimmed, and at the current duration
keepAudio	If non-zero, the request is to keep the audio in the trimmed result.
keepVideo	If non-zero, the request is to keep the video in the trimmed result.
newTrimIn	Return the acceptable in point of the trimmed clip. It must be at or before the requested in point.
newDuration	Return the acceptable duration. newTrimIn + newDuration must be at or after the trimIn + duration.
scale	The frame rate of the video clip, represented as scale over sampleSize.
sampleSize	

imCopyFileRec

Selector: *imCopyFile*

Describes how to copy a clip. Also provides a callback to update the progress bar and check if the user has cancelled.

```
typedef struct {
     void                *inPrivateData;
     csSDK_int32         *inPrefs;
     const prUTF16Char   *inSourcePath;
     const prUTF16Char   *inDestPath;
```

```
     importProgressFunc  inProgressCallback;
     void                *inProgressCallbackID;
} imTrimFileRec;
```

inPrivateData	Instance data gathered during *imGetInfo8* or *imGetPrefs8*.
inPrefs	Instance data gathered during *imGetPrefs8* (setup dialog).
inSourcePath	Full unicode path of the source file.
inDestPath	Full unicode path of the destination file.
inProgressCallback	importProgressFunc callback to call repeatedly to provide progress and to check for cancel by user. May be a NULL pointer, so make sure the function pointer is valid before calling.
inProgressCallbackID	Pass to progressCallback.

imDataRateAnalysisRec

Selector: *imDataRateAnalysis*

Specify the desired buffersize, return to Premiere with imNoErr; upon the next call fill buffer with imDataSamples, and specify a base data rate for audio (if any). This structure is used like imAnalysisRec.

```
typedef struct {
     void           *privatedata;
     void           *prefs;
     csSDK_int32    buffersize;
     char           *buffer;
     csSDK_int32    baserate;
} imDataRateAnalysisRec;
```

privatedata	Instance data gathered from *imGetInfo8* or *imGetPrefs8*.
prefs	Instance data gathered from *imGetPrefs8* (setup dialog info).
buffersize	The size of the buffer you request from Premiere prior to passing data back data in buffer.
buffer	Pointer to the analysis buffer to be filled with imDataSamples (see structure below).
baserate	Audio data rate (bytes per second) of the file.

```
typedef struct {
     csSDK_uint32 sampledur;
     csSDK_uint32 samplesize;
} imDataSample;
```

sampledur	Duration of one sample in video timebase, in samplesize increments; set the high bit if this is a keyframe.
samplesize	Size of this sample in bytes.

imDeferredProcessingRec

Selector: *imDeferredProcessing*

Describes the current progress of the deferred processing on the clip referred to by inPrivateData.

```
typedef struct {
     void            *inPrivateData;
     float           outProgress;
     char            outInvalidateFile;
     char            outCallAgain;
} imDeferredProcessingRec;
```

inPrivateData	Instance data gathered from *imGetInfo8* or *imGetPrefs8*.
outProgress	Set this to the current progress, from 0.0 to 1.0.
outInvalidateFile	The importer has updated information about the file.
outCallAgain	Set this to true to request that the importer be called again immediately.

imDeleteFileRec

Selector: *imDeleteFile*

Describes the file to be deleted.

```
typedef struct {
     csSDK_int32          filetype;
     const prUTF16Char    deleteFile;
} imDeleteFileRec;
```

filetype	The file's unique four character code, defined in the IMPT resource
deleteFile	Specifies the name (and path) of the file to be deleted.

imFileAccessRec8

Selectors: *imGetInfo8* and *imGetPrefs8*

Describes the file being imported.

```
typedef struct {
     void                 *importID;
     csSDK_int32          filetype;
     const prUTF16Char    *filepath;
     imFileRef            fileref;
     PrMemoryPtr          newfilename;
} imFileAccessRec;
```

importID	Unique ID provided by Premiere. Do not modify!
filetype	The file's unique four character code, defined in the IMPT resource.
filepath	The unicode file path and name.
fileref	A Windows HANDLE. Premiere does not overload this value by using it internally, although setting it to the constant kBadFileRef may cause Premiere to think the file is closed. This value is always valid.
newfilename	If the file is synthetic, the importer can specify the displayable name here as a prUTF16Char string during imGetPrefs8.

imFileAttributesRec

Selector: *imGetFileAttributes*

New in Premiere Pro 3.1. Provide the clip creation date.

```
typedef struct {
     prDateStamp    creationDateStamp;
     csSDK_int32    reserved[40];
} imFileAttributesRec;
```

creationDateStamp	Structure to store when the clip was created

imFileInfoRec8

Selector: *imGetInfo8*

The clip description sent to Premiere. Create and store any privateData.

```
typedef struct {
     char                 hasVideo;
     char                 hasAudio;
```

```
    imImageInfoRec      vidInfo;
    csSDK_int32         vidScale;
    csSDK_int32         vidSampleSize;
    csSDK_int32         vidDuration;
    imAudioInfoRec7     audInfo;
    PrAudioSample       audDuration;
    csSDK_int32         accessModes;
    void                *privatedata;
    void                *prefs;
    char                hasDataRate;
    csSDK_int32         streamIdx;
    char                streamsAsComp;
    prUTF16Char         streamName[256];
    csSDK_int32         sessionPluginID;
    char                alwaysUnquiet;
    char                highMemUsage;
    prUTF16Char         filePath[2048];
} imFileInfoRec8;
```

`hasVideo`	If non-zero, the file contains video.
`hasAudio`	If non-zero, the file contains audio.
`vidInfo`	If there is video in the file, fill out the `imImageInfoRec` structure (e.g. height, width, alpha info, etc.).
`vidScale` `vidSampleSize`	The frame rate of the video, represented as scale over sampleSize.
`vidDuration`	The total number of frames of video, in the video timebase.
`audInfo`	If there is audio in the file, fill out the `imAudioInfoRec7` structure.
`audDuration`	The total number of audio sample frames.
`accessModes`	The access mode of this file. Use one of the following constants: `kRandomAccessImport` - This file can be read by random access (default) `kSequentialAudioOnly` - When accessing audio, only sequential reads allowed (for variable bit rate files) `kSequentialVideoOnly` - When accessing video, only sequential reads allowed `kSequentialOnly` - Both sequential audio and video `kSeparateSequentialAudio` - Both random access and sequential access. This setting allows audio to be retrieved for scrubbing or playback even during audio conforming.
`privatedata`	Private instance data. Allocate a handle using Premiere's memory functions and store it here. Premiere will return the handle with subsequent selectors.

prefs	Settings data gathered from *imGetPrefs8* (setup dialog info).
hasDataRate	If set, the importer can read or generate data rate information for this file and will be sent *imDataRateAnalysis*.
streamIdx	The Premiere-specified stream index number
streamsAsComp	If multiple streams and this is stream zero, indicate whether to import as a composition or multiple clips.
streamName	Optional. The unicode name of this stream if there are multiple streams. New in Premiere Pro 3.1, an importer may use this to set the clip name based on metadata rather than the filename. The importer should set imImportInfoRec.canSupplyMetadataClipName to true, and fill out the name here.
sessionPluginID	This ID should be used in the File Registration Suite for registering external files (such as textures, logos, etc) that are used by an importer instance but do not appear as footage in the Project Window. Registered files will be taken into account when trimming or copying a project using the Project Manager. The sessionPluginID is valid only for the call that it is passed on.
alwaysUnquiet	Set to non-zero to tell Premiere if the clip should always be unquieted immediately when the application regains focus.
highMemUsage	Added in Premiere Pro CS3. Set to non-zero if this clip uses an extreme amount of memory. The number of files allowed to be open with this flag set to true is currently capped at 5.
filepath	Added in Premiere Pro 4.1. For clips that have audio in files separate from the video file, set the filename here, so that UMIDs can properly be generated for AAFs.

imFileOpenRec8

Selector: *imOpenFile8*

The file Premiere wants the importer to open.

```
typedef struct {
     imFileAccessRec8    fileinfo;
     void                *privatedata;
     csSDK_int32         reserved;
     PrFileOpenAccess    inReadWrite;
     csSDK_int32         inImporterID;
     csSDK_size_t        outExtraMemoryUsage;
} imFileOpenRec8;
```

fileinfo	imFileAccessRec8 describing the incoming file.

privatedata	Instance data gathered from *imGetInfo8* or *imGetPrefs8*.
reserved	Do not use.
inReadWrite	The file should be opened with the access mode specified: Either kPrOpenFileAccess_ReadOnly or kPrOpenFileAccess_ReadWrite
inImporterID	Can be used as the ID for calls in the PPix Cache Suite.
outExtraMemoryUsage	New in CS5. If the importer uses memory just by being open, which cannot otherwise be registered in the cache, put the size in bytes in this field.

imFileRef

Selectors: *imAnalysis*, *imDataRateAnalysis*, *imOpenFile8*, *imQuietFile*, *imCloseFile*, *imGetTimeInfo8*, *imSetTimeInfo8*, *imImportImage*, *imImportAudio7*

A file HANDLE on Windows, or a void* on MacOS. imFileRef is also a member of imFileAccessRec. Use OS-specific functions, rather than ANSI file functions, when manipulating imFileRef.

imFrameFormat

Selector: *imGetSourceVideo* (member of imSourceVideoRec)

Describes the frame dimensions and pixel format.

```
typedef struct {
     csSDK_int32    inFrameWidth;
     csSDK_int32    inFrameHeight;
     PrPixelFormat  inPixelFormat;
} imFrameFormat;
```

inFrameWidth	The frame dimensions requested.
inFrameHeight	
inPixelFormat	The pixel format of the frame requested.

imGetPrefsRec

Selector: *imGetPrefs8*

Contains preference data gathered from (or defaults to populate) a setup dialog. If you are creating media, use timelineData to walk through the current timeline in order to generate a preview of the media.

```
typedef struct {
     char            *prefs;
     csSDK_int32     prefsLength;
     char            firstTime;
     PrTimelineID    timelineData;
     void            *privatedata;
     TDB_TimeRecord tdbTimelineLocation;
     csSDK_int32     sessionPluginID;
     csSDK_int32     imageWidth;
     csSDK_int32     imageHeight;
     csSDK_uint32    pixelAspectNum;
     csSDK_uint32    pixelAspectDen;
     csSDK_int32     vidScale;
     csSDK_int32     vidSampleSize;
     float           sampleRate;
} imGetPrefsRec;
```

`prefs`	A pointer to a private structure (which you allocate) for storing dialog box preferences.
`prefsLength`	Prior to storing anything in the `prefs` member, set `prefsLength` to the size of your structure and return `imNoErr`; Premiere will re-size and call the plug-in again with *imGetPrefs8*.
`firstTime`	If set, *imGetPrefs8* is being sent for the first time. Instead, check to see if prefs has been allocated. If not, *imGetPrefs8* is being sent for the first time. Set up default values for the `prefsLength` buffer and present any setup dialog.
`timelineData`	Can be passed to `getPreviewFrameEx` callback along with `tdbTimelineLocation` to get a frame from the timeline beneath the current clip or timeline location. This is useful for titler plug-ins.
`privatedata`	Private instance data. Allocate a handle using Premiere's memory functions and store it here, if not already allocated in *imGetInfo8*. Premiere will return the handle with subsequent selectors.
`tdbTimelineLocation`	Can be passed to `getPreviewFrameEx` callback along with `timelineData` to get a frame from the timeline beneath the current clip or timeline location. This is useful for titler plug-ins.

sessionPluginID	This ID should be used in the File Registration Suite for registering external files (such as textures, logos, etc) that are used by a importer instance but do not appear as footage in the Project Window. Registered files will be taken into account when trimming or copying a project using the Project Manager. The sessionPluginID is valid only for the call that it is passed on.
imageWidth	New in CS5. The native resolution of the video.
imageHeight	
pixelAspectNum	New in CS5. The pixel aspect ratio of the video.
pixelAspectDen	
vidScale	New in CS5. The frame rate of the video, represented as scale over sampleSize.
vidSampleSize	
sampleRate	New in CS5. Audio sample rate.

imImageInfoRec

Selector: *imGetInfo8* (member of imFileInfoRec8)

Describes the video to be imported.

```
typedef struct {
     csSDK_int32     imageWidth;
     csSDK_int32     imageHeight;
     csSDK_uint16    pixelAspectV1;
     csSDK_uint16    depth;
     csSDK_int32     subType;
     char            fieldType;
     char            fieldsStacked;
     char            reserved_1;
     char            reserved_2;
     char            alphaType;
     matteColRec     matteColor;
     char            alphaInverted;
     char            isVectors;
     char            drawsExternal;
     char            canForceInternalDraw;
     char            dontObscure;
     char            isStill;
     char            noDuration;
     char            reserved_3;
     csSDK_uint32    pixelAspectNum;
     csSDK_uint32    pixelAspectDen;
```

```
    char            isRollCrawl;
    char            reservedc[3];
    csSDK_int32     importerID;
    csSDK_int32     supportsAsyncIO;
    csSDK_int32     supportsGetSourceVideo;
    csSDK_int32     hasPulldown;
    csSDK_int32     pulldownCadence;
    csSDK_int32     posterFrame;
    csSDK_int32     canTransform;
    csSDK_int32     interpretationUncertain;
    csSDK_int32     reserved[21];
} imImageInfoRec;
```

Plug-in Info	
`importerID`	Can be used as the ID for calls in the PPix Cache Suite.
`supportsAsyncIO`	Set this to true if the importer supports *imCreateAsyncImporter* and ai* selectors.
`supportsGet-SourceVideo`	Set this to true if the importer supports the *imGetSourceVideo* selector.
Bounds Info	
`imageWidth`	Frame width in pixels.
`imageHeight`	Frame height in pixels.
`pixelAspectNum`	The pixel aspect ratio numerator and denominator. For synthetic importers, these are by default the PAR of the project. Only set this if you need a specific PAR for the geometry of the synthesized footage to be correct.
`pixelAspectDen`	
Time Info	
`isStill`	If set, the file contains a single frame, so only one frame will be cached.

`noDuration`	One of the following:
	
	`imNoDurationFalse`
	`imNoDurationNoDefault`
	`imNoDurationStillDefault` - use the default duration for stills, as set by the user in the Preferences
	`imNoDurationNoDefault` - the importer will supply it's own duration
	
	This is primarily for synthetic clips, but can be used for importing non-sequential still images.

isRollCrawl	Set to non-zero value to specify this clip is a rolling or crawling title. This allows a player to optionally use the RollCrawl Suite to get sections of this title for real-time playback.
hasPulldown	New in Premiere Pro 2.0. Set this to true if the clip contains NTSC film footage with 3:2 pulldown.
pulldownCadence	New in Premiere Pro 2.0. Set this to the enumerated value that describes the pulldown of the clip: importer_PulldownPhase_NO_PULLDOWN 2:3 cadences importer_PulldownPhase_WSSWW importer_PulldownPhase_SSWWW importer_PulldownPhase_SWWWS importer_PulldownPhase_WWWSS importer_PulldownPhase_WWSSW 24pa cadences importer_PulldownPhase_WWWSW importer_PulldownPhase_WWSWW importer_PulldownPhase_WSWWW importer_PulldownPhase_SWWWW importer_PulldownPhase_WWWWS
posterFrame	New in Premiere Pro CS3. Poster frame number
Format Info	
depth	Bits per pixel. This currently has no effect and should be left unchanged.
subType	The four character code of the file's codec; associates files with MAL plug-ins. For uncompressed files, set to imUncompressed.
fieldType	One of the following: prFieldsNone prFieldsUpperFirst prFieldsLowerFirst prFieldsUnknown
fieldsStacked	Fields are present, and not interlaced.

alphaType	Used when depth is 32 or greater. One of the following: alphaNone - no alpha channel (the default) alphaStraight - straight alpha channel alphaBlackMatte - premultiplied with black alphaWhiteMatte - premultiplied with white alphaArbitrary - premultiplied with the color specified in matteColor alphaOpaque - alpha channel prefilled to opaque
matteColor	Newly used in Premiere Pro CS3. Used to specify matte color if alphaType is set to alphaArbitrary.
alphaInverted	If non-zero, alpha is treated as inverted (e.g. black becomes transparent).
canTransform	Set to non-zero value to specify this importer handles resolution independent files and can apply a transform matrix. The matrix will be passed during the import request in imImportImageRec.transform. This code path is currently not called by Premiere Pro. After Effects uses this call to import Flash video.
interpretationUncertain	Use an 'or' operator to combine any of the following flags: imPixelAspectRatioUncertain imFieldTypeUncertain imAlphaInfoUncertain
Unused	
pixelAspectV1	Maintain for backwards compatability. Plug-ins written for the Premiere 6.x or later API should use pixelAspectNum and pixelAspectDen.
isVectors	Use canTransform instead.
drawsExternal	
canForceInternalDraw	
dontObscure	

imImportAudioRec7

Selector: *imImportAudio7*

Describes the audio samples to be returned, and contains an allocated buffer for the importer to fill in. Provide the audio in 32-bit float, uninterleaved audio format.

```
typedef struct {
```

```
     PrAudioSample  position;
     csSDK_uint32   size;
     float          **buffer;
     void           *privatedata;
     void           *prefs;
} imImportAudioRec7;
```

`position`	In point, in audio sample frames. The importer should save the out point of the request in `privatedata`, because if position is less than zero, then the audio request is sequential, which means the importer should return contiguous samples from the last *imImportAudio7* call.
`size`	The number of audio sample frames to import.
`buffer`	An array of buffers, one buffer for each channel, with length specified in `size`. These buffers are allocated by the host application, for the plug-in to fill in with audio data.
`privatedata`	Instance data gathered from *imGetInfo8* or *imGetPrefs8*.
`prefs`	Instance data gathered from *imGetPrefs8* (setup dialog info).

imImportImageRec

Selector: *imImportImage*

Describes the frame to be returned.

```
typedef struct {
     csSDK_int32    onscreen;
     csSDK_int32    dstWidth;
     csSDK_int32    dstHeight;
     csSDK_int32    dstOriginX;
     csSDK_int32    dstOriginY;
     csSDK_int32    srcWidth;
     csSDK_int32    srcHeight;
     csSDK_int32    srcOriginX;
     csSDK_int32    srcOriginY;
     csSDK_int32    unused2;
     csSDK_int32    unused3;
     csSDK_int32    rowbytes;
     char           *pix;
     csSDK_int32    pixsize;
     PrPixelFormat  pixformat;
     csSDK_int32    flags;
     prFieldType    fieldType;
     csSDK_int32    scale;
```

```
    csSDK_int32     sampleSize;
    csSDK_int32     in;
    csSDK_int32     out;
    csSDK_int32     pos;
    void            *privatedata;
    void            *prefs;
    prRect          alphaBounds;
    csSDK_int32     applyTransform;
    float           transform[3][3];
    prRect          destClipRect;

} imImportImageRec;
```

Bounds Info	
dstWidth	Width of the destination rectangle (in pixels).
dstHeight	Height of the destination rectangle (in pixels).
dstOriginX	Origin X point (0 indicates the frame is drawn offscreen).
dstOriginY	Origin Y point (0 indicates the frame is drawn offscreen).
srcWidth	The same number returned as dstWidth.
srcHeight	The same number returned as dstHeight.
srcOriginX	The same number returned as dstOriginX.
srcOriginY	The same number returned as dstOriginY.
Frame Info	
rowbytes	The number of bytes in a single row of pixels.
pix	Pointer to a buffer into which the importer should draw. Allocated based on information from the *imGetInfo8*.
pixsize	The number of pixels. rowbytes * height.
pixformat	The pixel format Premiere requests.
flags	imDraftMode - Draw quickly if possible, using a faster and possibly less accurate algorithm. This may be passed when playing from the timeline. imSamplesAreFields - Most importers will ignore as Premiere already scales in/out/scale to account for fields, but if you need to know that this has occurred (because maybe you measure something in 'frames'), check this flag. Also, may we suggest considering measuring in seconds instead of frames?
fieldType	If the importer can swap fields, it should render the frame with the given field dominance: either imFieldsUpperFirst or imFieldsLowerFirst.
scale	The frame rate of the video, represented as scale over sampleSize.
sampleSize	

`in`	In point, based on the timebase defined by scale over sampleSize..
`out`	Out point, based on the timebase defined by scale over sampleSize..
`pos`	Import position, based on the above timebase. ***API bug***: Synthetic and custom importers will always receive zero. Thus, adjusting the in point on the timeline will not offset the in point.
`privatedata`	Instance data gathered during *imGetInfo* or *imGetPrefs*.
`prefs`	Instance data gathered during *imGetPrefs* (setup dialog info).
`alphaBounds`	New in Premiere Pro 1.5. This is the rect outside of which the alpha is always 0. Simply do not alter this field if the alpha bounds match the destination bounds. If set, the alpha bounds must be contained by the destination bounds. This is only currently used when a plug-in calls `ppixGetAlphaBounds`, and not currently used by any native plug-ins.
`applyTransform`	New in After Effects CS3. Not currently provided by Premiere. If non-zero, the host is requesting that the importer apply the transform specified in `transform` and `destClipRect` before returning the resulting image in `pix`.
`transform`	New in After Effects CS3. Not currently provided by Premiere. The source to destination transform matrix.
`destClipRect`	New in After Effects CS3. Not currently provided by Premiere. Destination rect inside the bounds of the pix buffer.

imImportInfoRec

Selector: *imInit*

Describes the importer's capabilities to Premiere.

```
typedef struct {
     csSDK_uint32   importerType;
     csSDK_int32    canOpen;
     csSDK_int32    canSave;
     csSDK_int32    canDelete;
     csSDK_int32    canResize;
     csSDK_int32    canDoSubsize;
     csSDK_int32    canDoContinuousTime;
     csSDK_int32    noFile;
     csSDK_int32    addToMenu;
     csSDK_int32    hasSetup;
     csSDK_int32    dontCache;
     csSDK_int32    setupOnDblClk;
     csSDK_int32    keepLoaded;
     csSDK_int32    priority;
```

```
    csSDK_int32    canAsync;
    csSDK_int32    canCreate;
    csSDK_int32    canCalcSizes;
    csSDK_int32    canTrim;
    csSDK_int32    avoidAudioConform;
    prUTF16Char    *acceleratorFileExt;
    csSDK_int32    canCopy;
    csSDK_int32    canSupplyMetadataClipName;
    csSDK_int32    reserved[20];
} imImportInfoRec;
```

Screen Info	
noFile	If set, this is a synthetic importer. The file reference will be zero.
addToMenu	If set to imMenuNew, the importer will appear in the File > New menu.
canDoContinuousTime	If set, the importer can render frames at arbitrary times and there is no set timecode. A color matte generator or a titler would set this flag.
canCreate	If set, Premiere will treat this synthetic importer as if it creates files on disk to be referenced for frames and audio. See Additional Details for more information on custom importers.
File handling flags	
canOpen	If set, the importer handles open and close operations. Set if the plug-in needs to be called to handle *imOpenFile*, *imQuietFile*, and *imCloseFile*.
canSave	If set, the importer handles File > Save and File > Save As after a clip has been captured and must handle the *imSaveFile* selector.
canDelete	If set, the importer can delete its own files. The plug-in must handle the *imDeleteFile* selector.
canCalcSizes	New in Premiere Pro 1.5. If set, the importer can calculate the disk space used by a clip during imCalcSize. An importer should support this call if it uses a tree of files represented as one top-level file to Premiere.
canTrim	New in Premiere Pro 1.5. If set, the importer can trim files during imTrimFile.
canCopy	New in Premiere Pro 2.0. Set this to true if the importer supports copying clips in the Project Manager.
Setup flags	
hasSetup	If set, the importer has a setup dialog. The dialog should be presented in response to *imGetPrefs*

setupOnDblClk	If set, the setup dialog should be opened whenever the user double clicks on a file imported by the plug-in the timeline or the project bin.
Memory handling flags	
dontCache	Unused.
keepLoaded	If set, the importer plug-in should never be unloaded. Don't set this flag unless it's absolutely necessary (it usually isn't).
Other	
priority	Determines priority levels for importers that handle the same filetype. Importers with higher numbers will override importers with lower numbers. Starting in Premiere Pro 1.0, more than 2 priority levels are now recognized. For overriding importers that ship with Premiere, use a value of 100 or greater. Higher-priority importers can defer files to lower-priority importers by returning imBadFile during *imOpenFile8* or *imGetInfo8*.
importType	Type identifier for the import module assigned based on the plug-in's IMPT resource. Do not modify this field.
avoidAudioConform	New in Premiere Pro 2.0. Set this to true if the importer supports fast audio retrieval and does not need the audio clips it imports to be conformed.
acceleratorFileExt	New in Premiere Pro 2.0. Fill this prUTF16Char array of size 256 with the file extensions of accelerator files that the importer creates and uses.
canSupplyMetadata-ClipName	New in Premiere Pro 3.1. Allows file based importer to set clip name from metadata. Set this in imFileInfoRec8.streamName.
Unused	
canResize	
canDoSubsize	
canAsync	

imIndFormatRec

Selector: *imGetIndFormat*

Describes the format(s) supported by the importer. Synthetic files can only have one format.

```
typedef struct {
     csSDK_int32    filetype;
     csSDK_int32    flags;
     csSDK_int32    canWriteTimecode;
```

```
    char            FormatName[256];
    char            FormatShortName[32];
    char            PlatformExtension[256];
    prBool          hasAlternateTypes;
    csSDK_int32     alternateTypes[kMaxAlternateTypes];
    csSDK_int32     canWriteMetaData;
} imIndFormatRec;
```

filetype	Unique four character code (fourcc) of the file.
flags	Legacy mechanism for describing the importer capabilities. Though the flags will still be honored for backward compatability, current and future importers should not use these flags. See table below for details.
canWriteTimecode	If set, timecode can be written for this filetype.
FormatName[256]	The descriptive importer name.
FormatShortName[256]	The short name for the plug-in, appears in the format menu.
PlatformExtension[256]	List of all the file extensions supported by this importer. If there's more than one, separate with null characters.
hasAlternateTypes	Unused
alternateTypes[kMaxAlternateTypes]	Unused
canWriteMetaData	New in 6.0. If set, imSetMetaData is supported for the filetype

The flags listed below are only for legacy plug-ins and should not be used.

Flag	Usage
xfIsMovie	Unused
xfCanReplace	Unused
xfCanOpen	Unused: Use imImportInfoRec.canOpen instead.
xfCanImport	Unused: The PiPL resource describes the file as an importer.
xfIsStill	Unused: Use imFileInfoRec.imImageInfoRec.isStill instead.
xfIsSound	Unused: Use imFileInfoRec.hasAudio instead.
xfCanWriteTimecode	If set, the importer can respond to imGetTimecode and imSetTimecode. Obsolete: use imIndFormatRec.canWriteTimecode instead.

`xfCanWriteMetaData`	Writes (and reads) metadata, specific to the importer's four character code filetype. Obsolete: use `imIndFormatRec.canWriteMetaData` instead.
`xfCantBatchProcess`	Unused

imIndPixelFormatRec

Selector: *imGetIndPixelFormat*

Describes the pixel format(s) supported by the importer.

```
typedef struct {
     void           *privatedata;
     PrPixelFormat  outPixelFormat;
} imIndPixelFormatRec;
```

`privatedata`	Instance data from *imGetInfo8* or *imGetPrefs8*.
`outPixelFormat`	One of the pixel formats supported by the importer

imMetaDataRec

Selector: *imGetMetaData* and *imSetMetaData*

Describes the metadata specific to a given four character code.

```
typedef struct {
     void           *privatedata;
     void           *prefs;
     csSDK_int32    fourCC;
     csSDK_uint32   buffersize;
     char           *buffer;
} imMetaDataRec;
```

`privatedata`	Instance data gathered during *imGetInfo8* or *imGetPrefs8*.
`prefs`	Instance data gathered during *imGetPrefs8* (setup dialog).
`fourcc`	Fourcc code of the metadata chunk.
`buffersize`	Size of the data in buffer.
`buffer`	The metadata.

imPeakAudioRec

Selector: *imGetPeakAudio*

Describes the peak values of the audio at the specified position.

```
typedef struct {
     void           *inPrivateData;
     void           *inPrefs;
     PrAudioSample  inPosition;
     float          inSampleRate;
     csSDK_int32    inNumSampleFrames;
     float          **outMaxima;
     float          **outMinima;
} imPeakAudioRec;
```

inPrivateData	Instance data gathered during *imGetInfo8* or *imGetPrefs8*.
inPrefs	Instance data gathered during *imGetPrefs8* (setup dialog).
inPosition	The starting audio sample frame of the peak data.
inSampleRate	The sample rate at which to generate the peak data.
inNumSampleFrames	The number of sample frames in each buffer.
outMaxima	An array of arrays to be filled with the maximum sample values.
outMinima	An array of arrays to be filled with the minimum sample values.

imPreferredFrameSizeRec

Selector: *imGetPreferredFrameSize*

Describes a frame size preferred by the importer.

```
typedef struct {
     void           *inPrivateData;
     void           *inPrefs;
     PrPixelFormat  inPixelFormat;
     csSDK_int32    inIndex;
     csSDK_int32    outWidth;
     csSDK_int32    outHeight;
} imPreferredFrameSizeRec;
```

inPrivateData	Instance data gathered during *imGetInfo8* or *imGetPrefs8*.
inPrefs	Instance data gathered during *imGetPrefs8* (setup dialog).
inPixelFormat	The pixel format for this preferred frame size.

inIndex	The index of this preferred frame size.
outWidth	The dimensions of this preferred frame size.
outHeight	

imQueryContentStateRec

Selector: *imQueryContentState*

Fill in the outContentStateID, which should be a GUID calculated based on the content state of the clip at inSourcePath. If the state hasn't changed since the last call, the GUID returned should be the same.

```
typedef struct {
     const prUTF16Char*  inSourcePath;
     prPluginID          outContentStateID;
} imQueryContentStateRec;
```

imQueryDestinationPathRec

Selector: *imQueryDestinationPath*

Fill in the desired outActualDestinationPath, based on the inSourcePath and inSuggestedDestinationPath.

```
typedef struct {
     void                *inPrivateData;
     void                *inPrefs;
     const prUTF16Char   *inSourcePath;
     const prUTF16Char   *inSuggestedDestinationPath;
     prUTF16Char         *outActualDestinationPath;
} imQueryDestinationPathRec;
```

inPrivateData	Instance data gathered during *imGetInfo8* or *imGetPrefs8*.
inPrefs	Instance data gathered during *imGetPrefs8* (setup dialog).
inSourcePath	The path of the source file to be trimmed
inSuggestedDestinationPath	The path suggested by Premiere where the destination file should be created.
outActualDestinationPath	The path where the importer wants the destination file to be created.

imSaveFileRec8

Selector: *imSaveFile8*

Describes the file to be saved.

```
typedef struct {
     void                *privatedata;
     csSDK_int32         *prefs;
     const prUTF16Char*  sourcePath;
     const prUTF16Char*  destPath;
     char                move;
     importProgressFunc  progressCallback;
     void                *progressCallbackID;
} imSaveFileRec8;
```

`privatedata`	Instance data gathered from *imGetInfo8* or *imGetPrefs8*.
`prefs`	Instance data gathered from *imGetPrefs8* (setup dialog info).
`sourcePath`	Full path of the file to be saved.
`destPath`	Full path the file should be saved to.
`move`	If non-zero, this is a move operation and the original file (the sourcePath) can be deleted after copying is complete.
`progressCallback`	Function to call repeatedly to provide progress and to check for cancel by user. May be a NULL pointer, so make sure the function pointer is valid before calling.
`progressCallbackID`	Pass to progressCallback.

imSourceVideoRec

Selector: *imGetSourceVideo*

Describes the requested frame, to be passed back in `outFrame`.

```
typedef struct {
     void           *inPrivateData;
     PrTime         inFrameTime;
     imFrameFormat  *inFrameFormats;
     csSDK_int32    inNumFrameFormats;
     bool           removePulldown;
     PPixHand       *outFrame;
     void           *prefs;
     csSDK_int32    prefsSize;
} imSourceVideoRec;
```

inPrivateData	Instance data gathered during *imGetInfo8* or *imGetPrefs8*.
inFrameTime	Time of frame requested.
inFrameFormats	An array of requested frame formats, in order of preference. If NULL, then any format is acceptable.
inNumFrameFormats	The number of frame formats in the inFrameFormats.
removePulldown	If true, pulldown should be removed if the pixel format supports it.
outFrame	Allocate memory to hold the requested frame, and pass it back here.
prefs	New in Premiere Pro 4.1. prefs data from *imGetPrefs8*
prefsSize	New in Premiere Pro 4.1. Size of prefs data.

imSubTypeDescriptionRec

Selector: *imGetSubTypeNames*

Added in Premiere Pro CS3. Describes the codec name associated with a given fourcc.

```
typedef struct {
     csSDK_int32     subType;
     prUTF16Char     subTypeName[256];
} imSubTypeDescriptionRec;
```

imTimeInfoRec8

Selector: *imGetTimeInfo8* and *imSetTimeInfo8*

Describes the timecode and timecode rate associated with a clip.

```
typedef struct {
     void            *privatedata;
     void            *prefs;
     char            orgtime[18];
     csSDK_int32     orgScale;
     csSDK_int32     orgSampleSize;
     char            alttime[18];
     csSDK_int32     altScale;
     csSDK_int32     altSampleSize;
     char            orgreel[40];
```

```
     char            altreel[40];
     char            logcomment[256];
     csSDK_int32     dataType;
} imTimeInfoRec;
```

privatedata	Instance data gathered during *imGetInfo8* or *imGetPrefs8*.
prefs	Instance data gathered during *imGetPrefs8* (setup dialog).
orgtime[18]	The original time in hours;minutes;seconds;frames, as captured from the source reel. The use of semi-colons indicates (to Premiere) drop-frame timecode, e.g. "00;00;00;00". Use colons for non-drop-frame timecode, e.g. "00:00:00:00".
orgScale	Timebase of the timecode in orgtime, represented as scale over sampleSize.
orgSampleSize	
alttime[18]	An alternative timecode (may differ from the source timecode), formatted as described above.
altScale	Timebase of the timecode in alttime.
altSampleSize	
orgreel[40]	Original reel name.
altreel[40]	Alternate reel name.
logcomment[256]	Comment string.
dataType	Currently always set to 1, denoting SMPTE timecode. More values may be added in the future.

imTrimFileRec8

Selector: *imTrimFile8*

Describes how to trim a clip, based on information returned by the importer during imCheckTrim8. Also provides a callback to update the progress bar and check if the user has cancelled.

```
typedef struct {
     void                *privatedata;
     void                *prefs;
     csSDK_int32         trimIn;
     csSDK_int32         duration;
     csSDK_int32         keepAudio;
     csSDK_int32         keepVideo;
     const prUTF16Char   *destFilePath;
     csSDK_int32         scale;
     csSDK_int32         sampleSize;
     importProgressFunc  progressCallback;
     void                *progressCallbackID;
```

```
} imTrimFileRec8;
```

privatedata	Instance data gathered during *imGetInfo8* or *imGetPrefs8*.
prefs	Instance data gathered during *imGetPrefs8* (setup dialog).
trimIn	In point of the trimmed clip, in the timebase specified by scale and sampleSize.
duration	Duration of the trimmed clip. If 0, then the request is to leave the clip untrimmed, and at the current duration
keepAudio	If non-zero, the request is to keep the audio in the trimmed result.
keepVideo	If non-zero, the request is to keep the video in the trimmed result.
destFilePath	The unicode path and name of the file to create.
scale	The frame rate of the video, represented as scale over sampleSize.
sampleSize	
progressCallback	importProgressFunc callback to call repeatedly to provide progress and to check for cancel by user. May be a NULL pointer, so make sure the function pointer is valid before calling.
progressCallbackID	Pass to progressCallback.

Suites

For information on how to acquire and manage suites, see the SweetPea Suites section.

Async File Reader Suite

New in Premiere Pro CS5. A cross-platform file handling suite.

Deferred Processing Suite

New in Premiere Pro 2.0. Allows an importer to schedule processing time when importing asynchronously, and to notify the user that the media item is pending additional processing. In the Project panel, the name of the item will be italicized, and its thumbnail will show as Pending.

Media Accelerator Suite

New in Premiere Pro 2.0. Allows an importer to register and find media accelerators (media metadata that improves performance, such as conformed audio files, audio waveform peak files, MPEG index files, etc).

In CS4, the suite is now at version 3. GetMediaAcceleratorDirectoryPath now takes an inOriginalPath. New GetDocumentIDAndContentState, which takes a path,

and returns a Document ID and Content State GUID, which are needed for which are needed for the register and find calls. `RegisterPathInDatabase` now requires a Document ID and Content State GUID as input. `FindPathInDatabase` now requires a Document ID and Content State GUID as input. It no longer requires the original media path.

6 Recorders

Recorders interface directly with capture hardware, and capture video and/or audio to any file format. The recorder is responsible for displaying any video preview in the Capture panel, playing any audio preview to the system sound or hardware output, driving the meters in the Audio Master Meters panel, providing any settings in a custom dialog, and digitizing the video and audio to a file (or multiple files) on disk. A recorder can optionally provide source timecode information to Premiere, and notify Premiere when the video format changes so that the capture preview area can be resized. The recorder does not communicate anything about the audio to Premiere. When recording is complete, Premiere imports the file using any available importers that support that filetype.

A recorder can only capture to a single filetype. To capture to several filetypes, provide several recorders.

If you've never developed a recorder before, you can skip the What's New sections, and go directly to Getting Started.

What's New?

What's New in Premiere Pro CS5?

Recorders can now display audio meters in the Audio Master Meters panel while previewing and capturing. Just use the new `AudioPeakDataFunc` callback passed in `recOpenParms` with *recmod_Open*.

What's New in Premiere Pro CS4?

Audio settings in the Capture Settings window are no longer supported. Any audio settings should be in the custom dialog shown by the recorder during *recmod_ShowOptions*. Recorders should no longer set the `acceptsAudioSettings` flag in `recInfoRec8`.

No More Project Presets

Since Premiere Pro CS4 support sequence-specific settings, project presets have been replaced by sequence presets. The difference from project presets are that sequence presets do not contain information to initialize capture settings. This means that users will have to initialize capture settings manually the first time they use a recorder with custom settings.

When the Capture Panel is invoked, if the capture settings are invalid or uninitialized, the Capture Settings dialog is invoked. In the Capture Settings dialog: If the recorder requires private data, the user cannot continue past the Capture Settings dialog until they have opened the current recorder's settings dialog.

What's New in Premiere Pro CS3?

A new function, `GetDeviceTimecodeFunc`, can be used to ask the device controller for its current timecode.

A new return value, `rmRequiresCustomPrefsError`, should be returned on *recmod_SetActive* if there are no valid capture prefs.

What's New in Premiere Pro 2.0?

New Selectors

recmod_GetSetupInfo8 - New selector for Unicode support, supercedes *recmod_GetSetupInfo*, passing `recCapSetups8` instead of `recCapSetups`.

recmod_PrepRecord8 - New selector for Unicode support, supercedes *recmod_PrepRecord*, passing `recCapParmsRec8` instead of `recCapParmsRec`.

recmod_Startup8 - New selector for Unicode support, supercedes *recmod_Startup*, passing `recInfoRec8` instead of `recInfoRec`.

New Structures

`recCapParmsRec8` - New structure for Unicode support, supercedes `recCapParmsRec`, using `recFileSpec8` instead of `recFileSpec` for the file path, and `SceneCapturedFunc8` instead of `SceneCapturedFunc`.

recCapSetups8 - New structure for Unicode support, supercedes recCapSetups, using recSetupItem8 instead of recSetupItem for the file path.

recFileSpec8 - New structure for Unicode support, supercedes recFileSpec, using a wchar_t[kPrMaxPath] instead of char[256] for the subtype name.

recInfoRec8 - New structure for Unicode support, supercedes recInfoRec, using wchar_t[255] instead of char[255] for recmodName.

New Callbacks

SceneCapturedFunc8 - New callback for Unicode support, supercedes SceneCapturedFunc, using wchar_t* instead of char* for inFileCaptured.

Getting Started

Selector Calling Sequence

The best ways to get familiar with the recorder API is to observe the messages sent between Premiere and the recorder plug-in.

recmod_Startup8 is sent once when Premiere launches. When Project Settings > Capture Settings is opened, *recmod_Open* is sent to create a new recorder instance and open the capture driver. *recmod_GetSetupInfo8* is then sent, so the recorder can specify which settings buttons (if any) should be enabled in the Capture Settings window when the recorder is selected.

If one or more settings buttons are enabled and then clicked by the user, *recmod_ShowOptions* is sent so the recorder can display a dialog (and save any user choices). When the Capture Settings window is closed, *recmod_Close* is sent to end the capture instance.

Whenever the Capture panel is open, the recorder will receive *recmod_SetActive* calls, with a parameter telling it to become active or inactive (based on user activity). *recmod_SetDisp* provides the plug-in the dimensions of the preview area in the Capture panel. *recmod_Idle* is repeatedly sent until the Record button is pushed, to give the recorder time to update the preview area and play audio coming from the capture hardware.

When the user clicks Record, or starts an In/Out or Batch capture, *recmod_PrepRecord8* is sent. The recorder prepares to capture, and if a start timecode is provided, tells the device controller to get the device into position using preRollFunc. The preRollFunc will block until the device is exactly in the right position, and when it returns, the recorder should immediately return

back to Premiere, open which *recmod_StartRecord* is then sent to the recorder, which should immediately starts capturing.

When the recorder starts capturing and returns from *recmod_StartRecord*, Premiere will repeatedly call *recmod_ServiceRecord* to give the recorder processor time. During recording, report status to Premiere with `StatusDispFunc`.

The capture may be stopped in several ways: The user could click the Stop button, the capture may reach the predetermined out point of an In/Out or Batch capture, or the recorder might return an error from *recmod_ServiceRecord*. In all cases, *recmod_StopRecord* will be sent to stop the capture, possibly followed by *recmod_CloseRecord* if there no more items in the batch. Finally, *recmod_Close* is sent when the Capture panel is closed to destroy the recorder instance.

recmod_Shutdown is sent when Premiere terminates.

Try the Sample Recorder Plug-in

Now that you've read the overview of the selector calling sequence above, build the sample recorder plug-in included with this SDK, and give it a whirl. To properly simulate a capture, you'll also need to create an .sdk media file and place it in the proper location.
1) Build the recorder, importer, and exporter into the plug-ins directory
2) Launch Premiere Pro and use the exporter to transcode any media file into the .sdk file format.
3) Place the newly created media file at "C:\premiere.sdk" on Windows, or "premiere.sdk" on the Desktop on Mac OS.
Now when you "capture" a file, it will use this file, and automatically import it using the importer.

Metadata

Pixel aspect ratio and timecode are provided by the recorder by filling out `recCapturedFileInfo`. Starting in CS4, after a clip has been captured, if Premiere has an XMP handler that supports the clip's filetype, the XMP handler will open the captured file and inject the information. If no such XMP handler is provided, the recorder is responsible for embedding any pixel aspect ratio information to the file, but Premiere will send *imSetTimeInfo8* to the importer to stamp the file with timecode.

Save Captured File Dialog

After a single clip is captured, the Save Captured File dialog allows the user to rename the filename of the clip just captured. The recorder is not involved in this process. Instead, the importer is called to open the newly captured clip, and it is sent *imSaveFile8* with the move flag set to true to move the file. This is handled by the importer, since *imSaveFile8* is usually already implemented to support the Project Manager.

Switching Preview Area Between Different Frame Sizes

`FormatChangedFunc` enables recorders to tell Premiere when the pixel aspect ratio has changed so the Capture Panel preview area can be resized. It can be called during preview, and even during capture.

Scene Detection

A recorder can optionally implement one or both of two features based on scene detection: Scene Capture and Scene Searching. What determines a scene break is up to the discretion of the recorder. The built-in DV recorder determines scene breaks by time/date breaks in the DV stream. But a recorder could analyze the video for breaks, or use any method it chooses to implement.

Scene Capture

Scene Capture enables a recorder to capture a continuous stream to multiple files divided up by scenes.

To support scene capture, the recorder must set `recInfoRec8.canCaptureScenes = kPrTrue` during *recmod_Startup8*. When the user captures with Scene Detect on, `recCapParmsRec8.captureScenes` will be non-zero during *recmod_PrepRecord8*. The recorder should begin capture, and when it detects the end of a scene, call `SceneCapturedFunc8`, to notify Premiere that a scene has been captured. Premiere passes back the `recFileSpec8` to give the recorder the filepath to which the next scene should be captured. Premiere also reserves memory for and passes back `recCapturedFileInfo` for the next capture.

Scene Searching

Scene Searching enables a recorder to fast forward or rewind to different scenes. The user can hit the Next Scene or Previous Scene buttons several times to seek several scenes away. Of course, this feature is only possible with the help of a device controller as well.

To support scene capture, the recorder must set `recInfoRec8.canSearchScenes = kPrTrue` during *recmod_Startup8*. When the user chooses Next Scene or Previous Scene, the recorder is sent *recmod_StartSceneSearch*. The scene searching algorithm happens in two passes. The first pass is a play fast forward or backward in the initial direction. In this mode, when the recorder passes the scene boundary, it should call `ReportSceneFunc` to tell Premiere the approximate range where the scene boundary is and return `rmEndOfScene`. Premiere will call *recmod_StopSceneSearch*, followed by *recmod_StartSceneSearch*, to start a new slow scan scene search in the opposite direction, passing back the approximate range reported by `ReportSceneFunc`. When the recorder reaches the scene boundary again, it should once again call `ReportSceneFunc` and return `rmEndOfScene`.

Entry Point

Below is the entry point function prototype for all recorder plug-ins. Premiere calls this entry point function to drive the recorder based on user input.

```
int RecEntry (
     csSDK_int32    selector,
     rmStdParms     *stdParms,
     void           *param1,
     void           *param2)
```

The *selector* is the action Premiere wants the recorder to perform. It tells the recorder the reason for the call. `stdParms` provides the recorder with callback functions to access additional information from Premiere or to have Premiere perform tasks. Parameters 1 and 2 contain state information and vary with the selector; they may contain a specific value or a pointer to a structure. Return `rmNoErr` if successful, or an appropriate return code.

Standard Parameters

This structure is sent from Premiere to the plug-in with every selector.

```
typedef struct {
     int                rmInterfaceVer;
     recCallbackFuncs   *funcs;
     piSuitesPtr        piSuites;
} rmStdParms;
```

Member	Description
`rmInterfaceVer`	Recorder API version Premiere Pro CS5 - `RECMOD_VERSION_10` Premiere Pro CS4 - `RECMOD_VERSION_9` Premiere Elements 3 - `RECMOD_VERSION_8` Premiere Pro CS3 - `RECMOD_VERSION_7`
`funcs`	Pointers to callbacks for recorders
`piSuites`	Pointer to universal callback suites

Recorder-Specific Callbacks

Recorders have access to ClassData Functions and Memory Functions through the `recCallbackFuncs`, which is a member of `rmStdParms`. `StatusDispFunc`, `PrerollFunc`,

ReportSceneFunc, and SceneCapturedFunc8 are accessible through recCapParmsRec8, which is sent with *recmod_PrepRecord8*.

```
typedef struct {
     ClassDataFuncsPtr   classFuncs;
     PlugMemoryFuncsPtr  memoryFuncs;
} recCallbackFuncs;

int (*StatusDispFunc){
     void *callbackID,
     char *stattext,
     int  framenum};

csSDK_int32 (*PrerollFunc)(
     void *callbackID);

void (*ReportSceneFunc)(
     void           *callbackID,
     csSDK_uint32   inSceneEdgeTimecode,
     csSDK_uint32   inEarliestSceneEdgeTimecode,
     csSDK_uint32   inGreatestSceneEdgeTimecode);

void (*SceneCapturedFunc8)(
     void                *callbackID,
     prUTF16Char         *inFileCaptured,
     recFileSpec8        *outNextSceneFilename,
     recCapturedFileInfo **outFileInfo);

void (*SceneCapturedFunc)(
     void                *callbackID,
     char                *inFileCaptured,
     recFileSpec         *outNextSceneFilename,
     recCapturedFileInfo **outFileInfo);

void (*FormatChangedFunc)(
     void           *callbackID,
     unsigned int   inPixelAspectRatioNum,
     unsigned int   inPixelAspectRatioDen,
     unsigned int   inMaxFrameWidth,
     unsigned int   inMaxFrameHeight,
     TDB_TimeRecord inFramerate,
     int            isDropFrame);

void (*GetDeviceTimecodeFunc)(
     void           *callbackID,
```

```
     csSDK_uint32    *outTimecode,
     TDB_TimeRecord *outFrameRate,
     int             *outIsDropFrame);

void (*AudioPeakDataFunc)(
     void                *callbackID,
     recAudioPeakData    *inAudioPeakData)
```

Function	Description
`classFuncs`	See ClassData functions
`memoryFuncs`	Legacy memory-related callbacks. These are the same ones passed in through piSuites.
`StatusDispFunc`	Available in `recCapParmsRec8` during *recmod_PrepRecord8*. Callback function pointer for use during capture to call into Premiere and update status information in the Capture Window. `callbackID` is the recording session instance passed in `recCapParmsRec`. `stattext` is text Premiere will display at the top of the Capture Window. `framenum` is the frame number being captured, represented in the absolute number of frames. For example, 00;00;04;03 in NTSC drop-frame timecode would be represented as 123.
`PrerollFunc`	Available in `recCapParmsRec8` during *recmod_PrepRecord8*. Callback function pointer to initiate device control pre-roll. Callback returns when the deck is playing at the proper frame. `callbackID` is the recording session instance passed in `recCapParmsRec`. Host returns a `prDevicemodError` to inform why the preroll failed.

`ReportSceneFunc`	Although this callback is obsolete for Scene Capture (superceded by `SceneCapturedFunc8`), it is still used for Scene Search to return the scene detected by the recorder. Available in `recSceneDetectionParmsRec` during *recmod_StartSceneSearch*. The `inSceneEdgeTimecode` parameter marks the timecode of the scene edge, if it can be determined exactly. If it cannot, it marks the approximated timecode of the edge, and the `inEarliestSceneEdgeTimecode` and `inGreatestSceneEdgeTimecode` parameters mark the earliest and latest possible timecodes that the scene would fall in between. If the scene break can be determined exactly, all three return parameters should be set to the same value.
`SceneCapturedFunc8`	New in Premiere Pro 2.0. Available in `recCapParmsRec8` during *recmod_PrepRecord8*. Callback to notify Premiere that a scene has been captured. Premiere returns the `recFileSpec8` to designate a filename for the next scene to capture and reserves memory for and returns `recCapturedFileInfo` for the next capture.
`SceneCapturedFunc`	Obsolete. Use `SceneCapturedFunc8` above.
`FormatChangedFunc`	Available in `recOpenParms` during *recmod_Open*. Use this when the pixel aspect ratio has changed so the Capture Panel can be resized. It can be called during preview, and even during capture.
`GetDeviceTimecodeFunc`	New in Premiere Pro CS3. Used to ask the device controller for its current timecode.

AudioPeakDataFunc	New in Premiere Pro CS5. Available in recOpenParms during *recmod_Open*. Use this to display audio meters in the Audio Master Meters panel while previewing and capturing. The values will be updated as long as the capture panel is active or front. This call can be made from any thread, at any time. Metering can be provided for up to 16 channels, in any configuration desired: 1, 2, 4, 6/5.1, 8, or 16. The recorder provides the peak amplitude in longAmplitude, and the current audio amplitude in shortAmplitude. The recorder can decide whether to pick a single value in longAmplitude, or do an average over the sound data. In Premiere Pro's built-in recorders, the long term peak data is currently buffered for 3 seconds at a time. If no new data is sent, it stays on the last value. So set the amplitude values to zero when finished.

Selector Table

This table summarizes the various selector commands a recorder can receive.

Selector	param1	param2
recmod_Startup8	recInfoRec8 *	unused
recmod_Shutdown	unused	unused
recmod_GetSetupInfo8	PrivateData	recCapSetups8
recmod_ShowOptions	PrivateData	recSetupParms
recmod_Open	PrivateData	recOpenParms
recmod_Close	PrivateData	unused
recmod_SetActive	PrivateData	(csSDK_int32)boolean toggle
recmod_SetDisp	PrivateData	recDisplayPos
recmod_Idle	PrivateData	recGetTimecodeRec
recmod_PrepRecord8	PrivateData	recCapParmsRec8
recmod_StartRecord	PrivateData	recCapturedFileInfo
recmod_ServiceRecord	PrivateData	unused
recmod_StopRecord	PrivateData	unused
recmod_CloseRecord	PrivateData	unused

recmod_StartSceneSearch		recSceneDetection-ParmsRec *
recmod_StopSceneSearch		unused
recmod_ServiceSceneSearch		unused
Obsolete - *recmod_Startup, recmod_GetSetupInfo, recmod_PrepRecord*		
Currently unused in CS4 - *recmod_QueryInfo, recmod_AudioIndFormat, recmod_GetAudioIndFormat7*, *recmod_StepRecord*, *recmod_StillRecord*		

Selector Descriptions

This section provides a brief overview of each selector and highlights implementation issues.

recmod_Startup8

```
param1 - recInfoRec8
param2 - unused
```

Sent once when Premiere launches so the plug-in can initialize and return its attributes to Premiere. The module should connect to any required capture hardware or drivers and fill in the `recInfoRec8`.

recmod_Shutdown

```
param1 - unused
param2 - unused
```

Sent when Premiere terminates. Deallocate any memory and release the capture hardware or driver.

recmod_GetSetupInfo8

```
param1 - PrivateData
param2 - recCapSetups8
```

Sent when the Capture Settings dialog is first displayed to obtain custom settings information. `recCapSetups8` provides text label fields button titles and enabling.

recmod_ShowOptions

```
param1 - PrivateData
```

```
param2 - recSetupParms
```

Sent when the user presses a settings button (one of four available) in the Capture Settings dialog. Request settings buttons during *recmod_GetSetupInfo8*.

`recSetupParms` indicates which button was pushed. If the `char *` passed in `recSetup-Parms` isn't NULL, it points to memory containing private data; otherwise, no previous settings are available. All the setup dialogs share the same memory; only one record is preserved. If there are several different setup records, they should all fit within one flattened memory allocation.

recmod_Open

```
param1 - PrivateData
param2 - recOpenParms
```

Sent when Premiere's New Project Settings > Capture Settings dialog or the Movie Capture window is displayed. Initialize hardware, create a private data structure for instance data, and pass a pointer to it back in param1. It will be sent back to you with subsequent selectors. `recOpen-Parms` contains information about the capture window and callbackID; store this information in private instance data.

recmod_Close

```
param1 - PrivateData
param2 - unused
```

Capture is complete and the capture window is closed. Disconnect from the hardware and deallocate the private instance data.

recmod_SetActive

```
param1 - PrivateData
param2 - boolean toggle
```

`param2` indicates whether the plug-in should activate. When a capture window is opened or receives the focus, it will be activated.

recmod_SetDisp

```
param1 - PrivateData
param2 - recDisplayPos
```

Sent when the capture window is resized or moved. Update a proxy or overlay in the capture window during capture. `recDisplayPos` specifies the new bounds. If they are unacceptable, modify them; the selector will be sent again with the new position. Set `mustresize` in `recDisplayPos` to resize the preview frame with the specified bounds. The plug-in is not allowed to resize the capture window, just the preview frame. If `mustresize` is set but the plug-in can't resize the frame, display something (black, grey, a graphic of your choice) for a preview. `mustresize` will be set when the Capture Settings dialog is being displayed.

recmod_Idle

```
param1 - PrivateData
param2 - recGetTimecodeRec *
```

Sent to give the plug-in processing time.

recmod_PrepRecord8

```
param1 - PrivateData
param2 - recCapParmsRec8
```

Set up for recording, based on the data in `recInfoRec8`. Use the `prerollFunc` callback to tell the device controller to get the device ready. Recording commences with the next selector, *recmod_StartRecord*.

If pressing the record button results in a recorder error before the *recmod_PrepRecord8* selector is even sent, make sure that the `fileType` four character code set in `recInfoRec8` is supported by an installed importer.

recmod_StartRecord

```
param1 - PrivateData
param2 - recCapturedFileInfo *
```

Sent after *recmod_PrepRecord*. Start capturing immediately. The pointer to `recCapturedFileInfo` is valid until the recording finishes.

recmod_ServiceRecord

```
param1 - PrivateData
param2 - unused
```

Sent repeatedly to give the plug-in processor time while recording.

recmod_StopRecord

```
param1 - PrivateData
param2 - unused
```

Stop recording and release record buffers.

recmod_CloseRecord

```
param1 - PrivateData
param2 - unused
```

Sent after *recmod_StopRecord*. During batch capturing, *recmod_StopRecord* will be called after every clip, but *recmod_CloseRecord* will not be called until after the last clip has been captured, to finalize the record process.

Return Codes

Return Code	Reason
`rmNoErr`	Operation has completed without error.
`rmUnsupported`	Unsupported command selector.
`rmAudioRecordError`	Audio recording error.
`rmVideoRecordError`	Video recording error.
`rmVideoDataError`	Data rate too high to record (return this if too many frames get dropped).
`rmDriverError`	Driver error.
`rmMemoryError`	Memory error.
`rmDiskFullError`	Disk full.
`rmDriverNotFound`	Can't connect to the capture driver.
`rmStatusCapture-Done`	Return from *recmod_StartRecord* when capture is complete.
`rmCaptureLimit-Reached`	Return from *recmod_ServiceRecord* when the (self-imposed) record limit time is reached.
`rmBadFormatIndex`	Invalid format index - stops *recmod_GetAudioIndFormat* queries from Premiere.
`rmFormatAccept`	The output format is valid.
`rmFormatDecline`	Cannot capture to this format.
`rmErrorPreroll-Abort`	Preroll function aborted.
`rmUserAbort`	Return from *recmod_StartRecord* if user aborts.

rmErrFileSize-LimitErr	Return from *recmod_ServiceRecord* if file size limit was reached.
rmFramesDropped	Return value to use for dropped frames.
rmDeviceRemoved	The device was removed during capture. Premiere assumes all material captured before this value as valid.
rmDeviceNotFound	The capture device was not found.
rmCapturedNoFrames	No frames were captured.
rmEndOfScene	If detecting scenes and recorder senses end of scene
rmNoFrameTimeout	Haven't seen any frames in a while, maybe the tape stopped or hit blank part of tape
rmCantDetect-ScenesError	If the recorder can't find the info it needs to properly judge scene bounds
rmCantFindRecord-InPoint	If capturing in to out and the recorder can't find the in point timecode
rmLastErrorSet	The recorder set the last error string using the SweetPea Error Suite
rmLastWarningSet	The recorder set the last warning string using the SweetPea Error Suite
rmLastInfoSet	The recorder set the last info string using the SweetPea Error Suite
rmIllegalAudio-FormatChange	Return when two different audio formats are recorded on a tape and the user tries to capture frames from both in a single capture.
rmRequiresCustom-PrefsError	New for Premiere Pro CS3. Return when no valid capture prefs are found during *recmod_SetActive*.
rmErrBadFile	Problem with output file.
rmIsCacheable	Return from *recmod_Startup8* if the plug-in is cacheable, rmNo-Error if not cacheable

Structures

Structure	Sent with selector
recInfoRec8	*recmod_Startup8*
recCapSetups8	*recmod_GetSetupInfo8*
recDisplayPos	*recmod_SetDisp*, *recmod_Open* (member of recOpenParms)
recOpenParms	*recmod_Open*
recCapturedFileInfo	*recmod_StartRecord*
recFileSpec8	*recmod_PrepRecord8* (member of recCapParmsRec8)

recSetupParms	*recmod_ShowOptions*
recCapParmsRec8	*recmod_PrepRecord8*
recGetTimecodeRec	*recmod_Idle*
recSceneDetectionParmsRec	*recmod_StartSceneSearch*
Obsolete - recInfoRec, recCapSetups, recFileSpec, recCapParmsRec	

Structure Descriptions

recInfoRec8

Selector: *recmod_Startup8*

Describes the recorder's capabilities to Premiere.

```
typedef struct {
     csSDK_int32    recmodID;
     csSDK_int32    fileType;
     csSDK_int32    classID;
     int            canVideoCap;
     int            canAudioCap;
     int            canStepCap;
     int            canStillCap;
     int            canRecordLimit;
     int            acceptsTimebase;
     int            acceptsBounds;
     int            multipleFiles;
     int            canSeparateVidAud;
     int            canPreview;
     int            wantsEvents;
     int            wantsMenuInactivate;
     int            acceptsAudioSettings;
     int            canCountFrames;
     int            canAbortDropped;
     int            requestedAPIVersion;
     int            canGetTimecode;
     int            reserved[16];
     csSDK_int32    prefTimescale;
     csSDK_int32    prefSamplesize;
     csSDK_int32    minWidth;
```

```
    csSDK_int32    minHeight;
    csSDK_int32    maxWidth;
    csSDK_int32    maxHeight;
    int            prefAspect;
    csSDK_int32    prefPreviewWidth;
    csSDK_int32    prefPreviewHeight;
    prUTF16Char    recmodName[256];
    csSDK_int32    audioOnlyFileType;
    int            canSearchScenes;
    int            canCaptureScenes;
    prPluginID     outRecorderID;
} recInfoRec, *recInfoPtr;
```

recmodID	Premiere's internal identifier for the plug-in. Never change this value.
fileType	Four character code for the captured file (for example 'AVIV' for Video for Windows .AVI files, and 'MOOV' for QuickTime .MOV files). Invent a unique code for proprietary formats as necessary, but make sure an importer is installed that supports the fourcc. If no such importer is installed, pressing the record button will result in a recorder error before the recmod_PrepRecord selector is even sent.
classID	Class identifier, used to differentiate between plug-ins that support the same fileType. ClassID is the identifying characteristic of plug-ins which form a media abstraction layer.
canVideoCap	If set, the recorder can capture video.
canAudioCap	If set, the recorder can capture audio
canStepCap	Unused
canStillCap	Unused
canRecordLimit	If set, the recorder can accepts recording time limits. The recorder will receive the user-specified record limit in recCapParmsRec.recordlimit. The plug-in must enforce the time limit during capture.
acceptsTimebase	If set, the recorder can capture to an arbitrary timebase.
acceptsBounds	If set, the recorder can capture to an arbitrary frame size.
multipleFiles	Unused
canSeparateVidAud	Unused

`canPreview`	Unused
`wantsEvents`	Unused
`wantsMenuInactivate`	Unused
`acceptsAudioSettings`	Unused, do not set
`canCountFrames`	If set, the recorder is expected to count frames and quit when the count is reached.
`canAbortDropped`	If set, the recorder can abort when frames are dropped
`requestedAPIVersion`	Unused
`canGetTimecode`	Can provide timecode from the capture stream (like DV).
`reserved[16]`	Do not use.
`activeDuringSetup`	If set, that the recorder shouldn't be deactivated before a *recmod_GetSetupInfo8* selector is issued
`prefTimescale`	Frames per second, in scale over sampleSize.
`prefSamplesize`	
`minWidth`	Define the minimum and maximum frame sizes the plug-in can capture. If the plug-in can only capture to a single fixed size, then set them to the same value.
`minHeight`	
`minWidth`	
`minHeight`	
`prefAspect`	Preferred frame aspect ratio for the captured frames. Shift the width into the high order word and the height into the low order word. For example, store 640x480 (a 4:3 aspect ratio) as: `prefAspect = (640 << 16) + 480;`
`prefPreviewWidth`	Unused
`prefPreviewHeight`	Unused
`recmodName[256]`	The recorder's name (appears in the Capture Format pulldown menu).
`audioOnlyFileType`	File type for audio-only captures. If 0, the video file type will be used.
`canSearchScenes`	If true, the recorder can detect a scene boundary for searching purposes
`canCaptureScenes`	If true, the recorder can identify when it has reached the end of a scene
`outRecorderID`	New in Premiere Pro 2.0. A GUID identifier is now required for all recorders. Editing Mode XMLs use these GUIDs to refer to recorders.

recCapSetups8

Selector: *recmod_GetSetupInfo8*

Enumerate custom setup buttons for the Capture Settings dialog, and pull-down menu items in the Capture panel.

```
typedef struct {
     int             customSetups;
     csSDK_int32     enableflags;
     recSetupItem8   setups[4];
} recCapSetups8;
```

customSetups	Number of setup buttons (up to 4).
enableflags	Bitstring where bits 0 to 3 correspond with setups 1 to 4. Set the appropriate bits to indicate to Premiere which setups should be enabled
setups[4]	Four recSetupItem8s used to label the setup buttons. A recSetupItem8 is just a prUTF16Char[256].

recDisplayPos

Selector: *recmod_SetDisp*, *recmod_Open* (member of recOpenParms)

Describes the display position for preview frames.

```
typedef struct {
     prWnd     wind;
     int       originTop;
     int       originLeft;
     int       dispWidth;
     int       dispHeight;
     int       mustresize;
} recDisplayPos;
```

wind	The window.
originTop	originTop and originLeft identify the offset in pixels from the top left of the window in which to display.
originLeft	

dispWidth	Display area dimensions.
dispHeight	
mustresize	If set, the video must be resized to fit within these bounds (see *recmod_SetDisp*).

recOpenParms

Selector: *recmod_Open*

Provides capture session information; save this information in private instance data.

```
typedef struct {
     recDisplayPos       disp;
     void                *callbackID;
     char                *setup;
     FormatChangedFunc   formatFunc;
     AudioPeakDataFunc   audioPeakDataFunc;
} recOpenParms;
```

disp	Preview display area
callbackID	Premiere's instance identifier for this recording session. Save this value for use with callback routines.
setup	If not null, points to settings saved from a previous recording session.
formatFunc	Use to inform Premiere of a new aspect ratio so the Capture panel can be updated
audioPeakDataFunc	New in CS5. Callback function to send audio metering data to be displayed by Premiere in the Audio Master Meters panel.

recCapturedFileInfo

Selector: *recmod_StartRecord*

Provide pixel aspect ratio and timecode of the captured file.

```
typedef struct {
     unsigned int   pixelAspectRatioNum;
     unsigned int   pixelAspectRatioDen;
     char           timeCode[31];
     TDB_TimeRecord tdb;
```

```
     char           date[31];
} recCapturedFileInfo;
```

pixelAspectRatioNum	Numerator of pixel aspect ratio.
pixelAspectRatioDen	Denominator of pixel aspect ratio.
timeCode	Text representation of timecode.
tdb	Timebase of the captured file.
date	New in Premiere Elements 7. The date of the the captured file, formatted in one of the following ways: “d/m/y” or “d/m/y h:m” or “d/m/y h:m:s”

recFileSpec8

Selector: *recmod_PrepRecord8* (member of `recCapParmsRec8`)

Used to describe the capture destination file.

```
typedef struct {
     short          volID;
     csSDK_int32    parID;
     prUTF16Char    name[kPrMaxPath];
} recFileSpec8;
```

volID	Unused
parID	Unused
name	Full file path.

recSetupParms

Selector: *recmod_ShowOptions*

Indicates which settings dialog should be displayed, and provides any previously saved settings.

```
typedef struct {
     uintptr_t parentwind;
     int       setupnum;
     char      *setup;
} recSetupParms;
```

parentwind	Parent window owner.
setupnum	Which setup button (1-4) was selected by the user.

setup	If not null, points to saved settings from previous sessions.

recCapParmsRec8

Selector: *recmod_PrepRecord8*

Specifies capture settings.

```
typedef struct {
     void                    *callbackID;
     int                     stepcapture;
     int                     capVideo;
     int                     capAudio;
     int                     width;
     int                     height;
     csSDK_int32             timescale;
     csSDK_int32             samplesize;
     csSDK_int32             audSubtype;
     csSDK_uint32            audrate;
     int                     audsamplesize;
     int                     stereo;
     char                    *setup
     int                     abortondrops;
     int                     recordlimit;
     recFileSpec8            thefile;
     StatusDispFunc          statFunc;
     PrerollFunc             prerollFunc;
     csSDK_int32             frameCount;
     char                    reportDrops;
     short                   currate;
     short                   timeFormat;
     csSDK_int32             timeCode;
     csSDK_int32             inHandleAmount;
     ReportSceneFunc         reportSceneFunc;
     int                     captureScenes;
     SceneCapturedFunc8      sceneCapturedFunc;
     bool                    recordImmediate;
     GetDeviceTimecodeFunc   getDeviceTimecodeFunc;
} recCapParmsRec8;
```

callbackID	Premiere's instance identifier for this recording session. Save this value for use with callback routines.
stepcapture	Unused
capVideo	If set, capture video.
capAudio	If set, capture audio.
width height	Dimensions of the video frames to capture. These are only sent if acceptsBounds was set in the recInfoRec. If the plug-in doesn't accept bounds, capture to the preferred dimensions we previously set in recInfoRec8.
timescale samplesize	Recording timebase. Only sent if acceptsTimebase was set in the recInfoRec8. Otherwise, capture using the timebase we previously set in recInfoRec8.
audSubtype audrate audsamplesize stereo	Unused
setup	Pointer to private instance data allocated in response to *recmod_GetSetupInfo8*.
abortondrops	If set, stop capture if frames are dropped.
recordlimit	Recording time limit, in seconds, only valid if canRecordLimit was set in recInfoRec8. Value passed in by Premiere. The plug-in must enforce the limit during capture.
thefile	Structure of type recFileSpec8 describing the capture destination file, only valid during recmod_PrepRecord8.
statFunc	Callback function pointer for use during capture to call into Premiere and update status information in the Capture Panel. See StatusDispFunc for more information.
preroll	Callback function pointer to initiate device control pre-roll. This callback is only initialized if it will be needed, meaning only it if doing an in/out capture or batch capture. Otherwise, this function pointer to be set to NULL. See PrerollFunc for more information.
frameCount	If canCountFrames was set in recInfoRec8, the number of frames to capture. No device polling will be done.

reportDrops	If non-zero, report dropped frames when they occur (by returning rmErrVidDataErr).
currate	Frames per second supported by the capture device (24, 25, 30).
timeFormat	0 = non-drop frame, 1 = drop frame timecode.
timeCode	Timecode for in-point of capture (-1 means ignore).
inHandleAmount	Number of frames of handle (buffered lead-in), previous to the user-specified capture in point, the record module requires.
reportSceneFunc	Obsolete. Use sceneCapturedFunc8 instead.
captureScenes	True if user has initiated scene capture
sceneCapturedFunc	Use this callback during scene capture to report the end of a scene
recordImmediate	If non-zero, begin recording immediately after device control returns from seek for pre-roll; don't wait for a timecode.
getDeviceTimecodeFunc	New for Premiere Pro CS3. Use this callback to ask the device controller for its current timecode.

recGetTimecodeRec

Selector: *recmod_Idle*

Allows the recorder to supply timecode information.

```
typedef struct {
     csSDK_int32    status;
     short          currate;
     short          timeFormat;
     csSDK_int32    timeCode;
     short          autoDetectDropness;
} recGetTimecodeRec;
```

status	0 indicates valid timecode, 1 indicates it's unknown or stale.
currate	30 for NTSC timecode, 25 for PAL.
timeFormat	0 for non-drop, 1 for drop-frame timecode.

timeCode	Timecode as an integer, represented in the absolute number of frames. For example, 00;00;04;03 in NTSC drop-frame timecode would be represented as 123.
autoDetectDropness	Non-zero if device controller has set DeviceRec.autoDetectDropness to true. This means that the device controller is relying on the recorder to determining whether the timecode is drop-frame or non-drop-frame. The recorder must call FormatChangedFunc if there is any change.

recSceneDetectionParmsRec

Selectors: *recmod_StartSceneSearch*

Used for scene searching. searchingForward is provided as a hint as the state of the device, and the reportSceneFunc should be used to notify Premiere of a scene boundary.

```
typedef struct {
     void                *callbackID;
     ReportSceneFunc     reportSceneFunc;
     int                 searchingForward;
     int                 searchMode;
     short               isDropFrame;
     csSDK_int32         earliestTimecode;
     csSDK_int32         greatestTimecode;
} recSceneDetectionParmsRec;
```

callbackID	Required for reportSceneFunc
reportSceneFunc	Use this to report the scenes
searchingForward	True if the tape is playing forward
searchMode	Either sceneSearch_FastScan or sceneSearch_SlowScan
isDropFrame	True if drop-frame, false otherwise
earliestTimecode	Only set for sceneSearch_SlowScan: in point for range to report scene edge
greatestTimecode	Only set for sceneSearch_SlowScan: out point for range to report scene edge

7 Exporters

Exporters are used to export video, audio, and markers in any format. Exporters can optionally provide hardware acceleration by coordinating with a renderer plug-in to render timeline segments. An exporter and a player combine to form an editing mode; the exporter generates preview files and the player manages the cutlist.

Exporters can be used from within Premiere Pro, or from Adobe Media Encoder. From within Premiere Pro, go to the File > Export > Media dialog. From there, the Export Settings dialog appears. The format chosen in the Format drop-down determines the exporter used, and the exporter provides the settings displayed in the Export Settings dialog.

What's New in CS5

`exQueryOutputFileListAfterExportRec` is now `exQueryOutputFileListRec`, with a slight change to the structure order.

We've also fixed a few bugs, such as bug 1925419, where all sliders would be given a checkbox to disable the control, as if `exParamFlag_optional` had been set.

3rd-party exporters can now be used to transcode assets to MPEG-2 or Blu-ray compliant files. Please refer to the Guidelines for Exporters in Encore for instructions on how to set up your exporter so that Encore can use it for transcoding.

Porting From the Compiler API

The export API replaces the old compiler API from CS3 and earlier versions. The export API combines the processing speed and quality of the old compiler API, with the UI flexibility of Media Encoder. Although the selectors and structures have been renamed and reorganized, much of the code that deals with rendering and writing frames is mostly the same.

The parameter UI is what has changed the most. Rather than having a standard set of parameters as standard compilers had, or having a completely custom UI as custom compilers had, in the new exporter API, all parameters must be explicitly added using the Export Param Suite. First register the parameters during *exSelGenerateDefaultParams*, and then provide the localized strings and constrained parameter values during *exSelPostProcessParams*. When the exporter is sent *exSelExport* to export, get the parameter values, again using the Export Param Suite.

Getting Started

Start your plug-in by modifying one of the SDK samples. Step through the code in your debugger to learn the order of events.

Multiple File Formats

To support more than one file format in a single exporter, describe one format at a time during *exSelStartup*. After describing the first one, return `exportReturn_IterateExporter` from *exSelStartup*, and the exporter will be called again to describe the second format, and so on. After describing the last format, return `exportReturn_IterateExporter`, and the exporter will be called yet again. This time, return `exportReturn_IterateExporterDone`.

Use a unique `fileType` for each format. When you are later sent *exSelGenerateDefaultParams*, *exSelPostProcessParams*, etc, you'll want to pay attention to the `fileType`, and respond according to the format.

Adding Parameters

Add parameters using the Export Param Suite. First register the parameters during *exSelGenerateDefaultParams*, and then provide the localized strings and constrained parameter values during *exSelPostProcessParams*. When the exporter is sent *exSelExport* to export, get the parameter values, again using the Export Param Suite.

Media Encoder as a Test Harness

Adobe Media Encoder can be launched as a separate application. It may be faster to developing exporters using Media Encoder, since it is a lighter-weight application. However, you will want to test your exporter in Premiere Pro, to make sure the behavior is the same as when running in Media Encoder.

Creating Presets

Create your own presets using the Export Settings UI, either from within Premiere Pro, or Media Encoder. Just modify the parameters the way you want, and hit the Save icon to save the preset to disk. The presets are saved with the extension '.epr'.

Starting in CS5, all the presets are saved to the same location, regardless of whether saved from Premiere Pro or Media Encoder:

On Windows Vista 64, presets are saved here:
`[User folder]\AppData\Roaming\Adobe\Common\AME\[version]\Presets\`

In CS4, where the files are saved depends on whether you've opened the Export Settings UI in Premiere Pro or Media Encoder:

Media Encoder presets
On Windows Vista, presets are saved here:
`[User folder]\AppData\Roaming\Adobe\Adobe Media Encoder\[version]\Presets\`
On Windows XP:
`[Documents and Settings folder]\[user name]\Application Data\Adobe\Adobe Media Encoder\[version]\Presets\`
On Mac OS:
`Macintosh HD/Users/[user name]/Library/Preferences/Adobe/Adobe Media Encoder/[version]/Presets/`

Premiere Pro presets
On Windows Vista, presets are saved here:
`[User folder]\AppData\Roaming\Adobe\Premiere Pro\[version]\Presets\`
On Windows XP:
`[Documents and Settings folder]\[user name]\Application Data\Adobe\Premiere Pro\[version]\Presets\`
On Mac OS:
`Macintosh HD/Users/[user name]/Library/Preferences/Adobe/Adobe Premiere Pro/[version]/Presets/`

For better performance, we recommend you install any presets for your exporter in the application folder for Premiere Pro and Media Encoder. For both Windows and Mac OS:
`[App installation path]\MediaIO\systempresets\[exporter subfolder]`
The subfolder must be named based on the hexadecimal fourCCs of the `ClassID` and `filetype` of the exporter. For example, the SDK exporter has a `ClassID` of `'DTEK'` or `0x4454454B`, and a filetype of `'SDK_'` or `0x53444B5F`. So the subfolder must be named

'4454454B_53444B5F'. For convenience, you can find the ClassID and filetype fourCCs in the preset file itself, in a decimal representation.

Parameter Caching

During development, when you modify parameters in your exporter and reload the plug-in into the host, the Settings UI may continue to show stale parameter data. New parameters that you have added may not appear, or old ones may continue to appear. Or if you have changed the UI for an existing parameter, it may not take effect.

At a minimum, any old presets must be deleted. This includes Media Encoder presets and Premiere Pro presets. After deleting the old presets, there are two options, depending on whether the an older version of the exporter has already been distributed and is in use.

Increment the Parameter Version

If an older version of the exporter is already being used by customers, you'll need to use parameter versioning. During *exSelGenerateDefaultParams*, you should call SetParamsVersion() in the Export Param Suite and increment the version number.

After that, create new presets and sequence encoder presets (if needed) using the new set of parameters. Make sure your installer removes the old presets, and installs the new ones.

Flush the Parameter Cache

If you don't increment the parameter version, you can manually flush the parameter cache in a few steps. After you've deleted the old presets, do the following:

1) Delete hidden presets that were created by the hosts for the most recently used parameter settings. Look for a file called Placeholder Preset.epr in both the folders above the Media Encoder presets and the Premiere Pro presets.
2) Delete batch.xml, used by Media Encoder. This is also in the folder above the Media Encoder presets. Deleting this is equivalent to deleting the items out of the Media Encoder render queue.
3) Delete Premiere Pro sequence encoder presets that use the exporter, if any
4) Even after deleting all the old presets, Media Encoder may initially show old cached parameter UI. In the Settings UI, just switch to a different format and then back to yours.

Exporters Used for Editing Modes

Any exporter that is used in an editing mode must have a codec parameter, and that parameter ID must be `ADBEVideoCodec`. If Premiere Pro cannot find this parameter, it will not be able to reopen projects in the custom editing mode, and will revert the project to Desktop mode.

Sequence Encoder Presets

Sequence preview presets are now required for editing modes. These contain the exporter parameters to generate preview files. This makes preview file formats much easier to define, by using the Media Encoder or Premiere Pro UI to create presets, rather than directly editing XML.

To create a sequence encoder preset:
1) Create a preset. The name that you give it will be the name that will be used in the Sequence Settings > General > Preview File Format drop-down.
2) Make sure this preset is installed in the application folder for Premiere Pro, along with the other sequence presets:

On Windows, they should be installed here:
[App installation path]\Settings\EncoderPresets\SequencePreview\[editing mode GUID]*.epr
On MacOS, it is basically the same (inside the application package):
[App installation path]/[Premiere Pro package]/Contents/Settings/EncoderPresets/SequencePreview/[editing mode GUID]/*.epr

As you can see by the installation paths above, Premiere Pro associates the sequence preview presets with the editing mode they go with, by using the presets in the folder that matches the GUID of the editing mode. The editing mode GUID is defined in the editing mode XML file, using the `<EditingMode.ID>` tag.

You can not only provide sequence preview presets for your own editing mode, but you could even add additional sequence preview presets for one of the built-in editing modes. Editing mode GUIDs for built-in editing modes can be found in the `Adobe Editing Modes.xml` file. For example, the Desktop editing mode on Windows has the GUID `9678AF98-A7B7-4bdb-B477-7AC9C8DF4A4E`. On Mac OS it is `795454D9-D3C2-429d-9474-923AB13B7018`.

Timeline Segments in Exporters

The timeline segments available to exporters do not always fully describe the sequence being exported. To consistently get timeline segments that fully describe the sequence, an exporter needs to work along with a renderer plug-in.

During a sequence export, Premiere Pro makes a copy of the project file and passes it to Media Encoder. Media Encoder takes that project and uses the PProHeadless process to generate rendered frames. So when an exporter, which is running in Media Encoder, parses the sequence, it only has a very high-level view. It sees the entire sequence as a single clip, and sees any optional cropping or filters as applied effects. So when parsing that simple, high-level sequence, if there are no effects, an exporter can just use the `MediaNode`'s `ClipID` with the Clip Render Suite to get frames directly from the PProHeadless process. In the PProHeadless process, a renderer plug-in can step in, parse the real sequence in all its glory, and optionally provide frames in a custom pixel format.

When rendering preview files, Premiere Pro does the rendering without Media Encoder, so an exporter can get the individual segments for each clip, similar to before.

Smart Rendering

Under very specific circumstances, an exporter can request compressed frames, avoiding unnecessary de/recompression. This would be done by providing both exporter and renderer plug-ins that parse timeline segments. If the source can be copied over to the destination, the compressed frames can be passed in a custom pixel format. These compressed frames are not guaranteed, however, so the exporter should be prepared to handle uncompressed frames.

Entry Point

```
DllExport PREMPLUGENTRY xSDKExport (
     csSDK_int32          selector,
     exportStdParms*      stdParmsP,
     void*                param1,
     void*                param2)
```

selector is the action the host wants the exporter to perform. `stdParms` provides callbacks to obtain additional information from the host or to have the host perform tasks. Parameters 1 and 2 vary with the selector; they may contain a specific value or a pointer to a structure. Return `exportReturn_ErrNone` if successful, or an appropriate return code.

Standard Parameters

A pointer to this structure is sent from the host to the plug-in with every selector. See Universals.

```
typedef struct {
     csSDK_int32               interfaceVer;
     plugGetSPBasicSuiteFunc* getSPBasicSuite;
} exportStdParms;
```

Member	Description
`interfaceVer`	Exporter API version Premiere Pro CS5 - `prExportVersion200` Premiere Pro 4.0.1 through 4.2.1 - `prExportVersion101` Premiere Pro CS4 - `prExportVersion100`
`getSPBasicSuite`	This very important call returns the SweetPea suite that allows plug-ins to acquire and release all other SweetPea suites. `SPBasicSuite* getSPBasicSuite();`

Selector Table

This table summarizes the various selector commands an exporter can receive.

Selector	param1	param2
exSelStartup	`exExporterInfoRec`*	unused
exSelBeginInstance	`exExporterInstanceRec`*	unused
exSelGenerateDefaultParams	`exGenerateDefaultParamRec`*	unused
exSelPostProcessParams	`exPostProcessParamsRec`*	unused
exSelValidateParamChanged	`exParamChangedRec`*	unused
exSelGetParamSummary	`exParamSummaryRec`*	unused
exSelParamButton	`exParamButtonRec`*	unused
exSelExport	`exDoExportRec`*	unused
exSelQueryExportFileExtension	`exQueryExportFileExtensionRec`*	unused
exSelQueryOutputFileList	`exQueryOutputFileList`*	unused
exSelQueryStillSequence	`exQueryStillSequenceRec`*	unused
exSelQueryOutputSettings	`exQueryOutputSettingsRec`*	unused
exSelValidateOutputSettings	`exValidateOutputSettingsRec`*	unused
exSelEndInstance	`exExporterInstanceRec`*	unused
exSelShutdown	unused	unused

Selector Descriptions

This section provides a brief overview of each selector and highlights implementation issues. Additional implementation details are at the end of the chapter.

exSelStartup

```
param1 - exExporterInfoRec *
param2 - unused
```

Sent during application launch, unless the exporter has been cached. A single exporter can support multiple codecs and file extensions. `exExporterInfoRec` describes the exporter's attributes, such as the format display name.

exSelBeginInstance

```
param1 - exExporterInstanceRec *
param2 - unused
```

Allocate any private data.

exSelGenerateDefaultParams

```
param1 - exGenerateDefaultParamRec *
param2 - unused
```

Set the exporter's default parameters using the Export Param Suite.

exSelPostProcessParams

```
param1 - exPostProcessParamsRec *
param2 - unused
```

Post process parameters. This is where the localized strings for the parameter UI must be provided.

exSelValidateParamChanged

```
param1 - exParamChangedRec *
param2 - unused
```

Validate any parameters that have changed. The exporter may correct an invalid combination of parameters here, or dim/undim certain parameter controls based on the current settings.

exSelGetParamSummary

```
param1 - exParamSummaryRec *
param2 - unused
```

Provide a text summary of the current parameter settings, which will be displayed in the summary area of the Export Settings dialog.

exSelParamButton

```
param1 - exParamButtonRec *
param2 - unused
```

Sent if exporter has one or more buttons in its parameter UI, and the user clicks one of the buttons in the Export Settings. The ID of the button pressed is passed in `exParamButtonRec.buttonParamIdentifier`. Display any dialog using platform-specific UI, collect any user input, and save any changes back to `privateData`.

exSelExport

```
param1 - exDoExportRec *
param2 - unused
```

Do the export! Sent when the user starts an export to the format supported by the exporter, or if the exporter is used in an Editing Mode and the user renders the work area.

Single file exporters are sent this selector only once per export (e.g. AVI, QuickTime). To create a single file, setup a loop where you request each frame in the `startTime` to `endTime` range using one of the render calls in the Sequence Render Suite and `GetAudio` in the Sequence Audio Suite. For better performance, you can use the asynchronous calls in the Sequence Render Suite to render multiple frames on multiple threads.

Still frame exporters are sent *exSelExport* for each frame in the sequence (e.g. numbered TIFFs). The host will name the files appropriately.

Save render time by checking to see if frames are repeated. Inspect the `SequenceRender_GetFrameReturnRec.repeatCount` returned from a render call, which holds a frame repeat count.

exSelQueryExportFileExtension

```
param1 - exQueryExportFileExtensionRec *
```

```
param2 - unused
```

For exporters that support more than one file extension, specify an extension given the file type. If this selector is not supported by the exporter, the extension is specified by the exporter in `exExporterInfoRec.fileTypeDefaultExtension`.

exSelQueryOutputFileList

```
param1 - exQueryOutputFileListRec *
param2 - unused
```

For exporters that export to more than one file. This is called before an export for the host to find out which files would need to be overwritten. It is called after an export so the host will know about all the files created, for any post encoding tasks, such as FTP. If this selector is not supported by the exporter, the host application will only know about the original path.

This selector will be called three times. On the first call, the plug-in fills out `numOutputFiles`. The host will then make `numOutputFiles` count of `outputFileRecs`, but empty. On the second call, the plug-in fills out the path length (incl trailing null) for each `exOutputFileRec` element in `outputFileRecs`. The host will then allocate all paths in each `outputFileRec`. On the third call, the plug-in fills in the path members of the `outputFileRecs`.

exSelQueryStillSequence

```
param1 - exQueryStillSequenceRec *
param2 - unused
```

The host application asks a still-only exporter if it wants to export as a sequence, and at what frame rate.

exSelQueryOutputSettings

```
param1 - exQueryOutputSettingsRec *
param2 - unused
```

The host application asks the exporter for general details about the current settings. This is a required selector.

exSelValidateOutputSettings

```
param1 - exValidateOutputSettingsRec *
param2 - unused
```

The host application asks the exporter if it can export with the current settings. The exporter should return `exportReturn_ErrLastErrorSet` if not, and the error string should be set to a description of the failure.

exSelEndInstance

```
param1 - exExporterInstanceRec *
param2 - unused
```

Deallocate any private data.

exSelShutdown

```
param1 - unused
param2 - unused
```

Sent immediately before shutdown. Free all remaining memory and close any open file handles.

Return Codes

Return Code	Reason
`exportReturn_ErrNone`	Operation has completed without error.
`exportReturn_Abort`	User aborted the export.
`exportReturn_Done`	Export finished normally.
`exportReturn_InternalError`	Return this if none of the other errors apply.
`exportReturn_OutOfDiskSpace`	Out of disk space error.
`exportReturn_BufferFull`	The offset into the buffer would overflow it.
`exportReturn_ErrOther`	The vaguer the better, right?
`exportReturn_ErrMemory`	Out of memory.
`exportReturn_ErrFileNotFound`	File not found.
`exportReturn_ErrTooManyOpenFiles`	Too many open files.
`exportReturn_ErrPermErr`	Permission violation.
`exportReturn_ErrOpenErr`	Unable to open the file.
`exportReturn_ErrInvalidDrive`	Invalid drive.
`exportReturn_ErrDupFile`	Duplicate filename.
`exportReturn_ErrIo`	File I/O error.
`exportReturn_ErrInUse`	File is in use.

exportReturn_IterateExporter	Return value from *exSelStartup* to request exporter iteration.
exportReturn_IterateExporterDone	Return value from *exSelStartup* to indicate there are no more exporters.
exportReturn_InternalErrorSilent	Return error code from *exSelExport* to put a custom error message on screen just before returning control to the host.
exportReturn_ErrCodecBadInput	A video codec refused the input format.
exportReturn_ErrLastErrorSet	The exporter is returning an error using the Error Suite.
exportReturn_ErrLastWarningSet	The exporter is returning a warning using the Error Suite.
exportReturn_ErrLastInfoSet	The exporter is returning information using the Error Suite.
exportReturn_ErrExceedsMaxFormatDuration	The exporter (or the host) has deemed the duration of the export to be too large.
exportReturn_VideoCodecNeedsActivation	The current video codec is not activated and cannot be used.
exportReturn_AudioCodecNeedsActivation	The current audio codec is not activated and cannot be used.
exportReturn_IncompatibleAudioChannelType	The requested audio channels are not compatible with the source audio.
exportReturn_IncompatibleVideoCodec	New in CS5. User tried to load a preset with an invalid video codec
exportReturn_IncompatibleAudioCodec	New in CS5. User tried to load a preset with an invalid audio codec
exportReturn_Unsupported	Unsupported selector.

Structures

Structure	Sent with selector
exDoExportRec	*exSelExport*
exExporterInfoRec	*exSelStartup*
exExporterInstanceRec	*exSelBeginInstance* and *exSelEndInstance*
exGenerateDefaultParamRec	*exSelGenerateDefaultParams*
exParamButtonRec	*exSelParamButton*
exParamChangedRec	*exSelValidateParamChanged*
exParamSummaryRec	*exSelGetParamSummary*
exPostProcessParamsRec	*exSelPostProcessParams*

exQueryExportFileExtensionRec	*exSelQueryExportFileExtension*
exQueryOutputFileListRec	*exSelQueryOutputFileList*
exQueryOutputSettingsRec	*exSelQueryOutputSettings*
exQueryStillSequenceRec	*exSelQueryStillSequence*
exValidateOutputSettingsRec	*exSelValidateOutputSettings*

Structure Descriptions

exDoExportRec

Selector: *exSelExport*

Provides general export settings. The exporter should retrieve the parameter settings from the Export Param Suite.

```
typedef struct {
     csSDK_uint32        exporterPluginID;
     void*               privateData;
     csSDK_uint32        fileType;
     csSDK_int32         exportAudio;
     csSDK_int32         exportVideo;
     PrTime              startTime;
     PrTime              endTime;
     csSDK_uint32        fileObject;
     PrTimelineID        timelineData;
     csSDK_int32         reserveMetaDataSpace;
     csSDK_int32         maximumRenderQuality;
} exDoExportRec;
```

exporterPluginID	The host's internal identifier for this exporter, used for various suite calls, such as in the Sequence Render Suite and Sequence Audio Suite.
privateData	Data allocated and managed by the exporter.
fileType	The file format four character code set by the exporter during *exSelStartup*. Indicates which format the exporter should write, since exporters can support multiple formats.
exportAudio	If non-zero, export audio.
exportVideo	If non-zero, export video.
startTime	The start time of the sequence to export.

endTime	The end time of the sequence to export. If startTime is 0, also the total durection to export. Range specified is [startTime, endTime), meaning the endTime is not actually included in the range.
fileObject	For use with the Export File Suite, to get and manipulate the file specified by the user.
timelineData	Handle used for the Timeline Functions.
reserveMeta-DataSpace	Amount to reserve in a file for metadata storage.
maximumRenderQual-ity	If non-zero, render at the highest possible quality.

exExporterInfoRec

Selector: *exSelStartup*

Describe the exporter's capabilities by filling out this structure. For each filetype, populate exExporterInfoRec and return exportReturn_IterateExporter. *exSelStartup* will then be resent. Repeat the process until there are no more file formats to describe, then return exportReturn_IterateExporterDone. The fileType indicates which format the exporter should currently work with in subsequent calls.

```
typedef struct {
     csSDK_uint32   unused;
     csSDK_uint32   fileType;
     prUTF16Char    fileTypeName[256];
     prUTF16Char    fileTypeDefaultExtension[256];
     csSDK_uint32   classID;
     csSDK_int32    exportReqIndex;
     csSDK_int32    wantsNoProgressBar;
     csSDK_int32    hideInUI;
     csSDK_int32    doesNotSupportAudioOnly;
     csSDK_int32    canExportVideo;
     csSDK_int32    canExportAudio;
     csSDK_int32    singleFrameOnly;
     csSDK_int32    maxAudiences;
     csSDK_int32    interfaceVersion;
     csSDK_uint32   isCacheable;
} exExporterInfoRec;
```

fileType	The file format four character code (e.g. 'AVIV' = Video for Windows, 'MooV' = QuickTime).
fileTypeName	The localized display name for the fileype.

fileTypeDefaultExtension	The default extension for the filetype. An exporter can support multiple extensions per filetype, by implementing *exSelQueryExportFileExtension*.
classID	Class identifier for the module, differentiates between exporters that support the same filetype and creates associations between different Media Abstraction Layer plug-ins.
exportReqIndex	If an exporter supports multiple filetypes, this index will be incremented by the host for each call, as the exporter is requested to describe its capabilities for each filetype. Initially zero, incremented by the host each time the exporter returns exportReturn_IterateExporter.
wantsNoProgressBar	If non-zero, the default exporter progress dialog will be turned off, allowing the exporter to display its own progress dialog. The exporter also will not get exportReturn_Abort errors from the host during callbacks – it must detect an abort on its own, and return exportReturn_Abort from *exSelExport* if the user aborts the export.
hideInUI	Set this to non-zero if this filetype should only be used for making preview files, and should not be visible as a general export choice.
doesNotSupportAudioOnly	Set this to non-zero for filetypes that do not support audio-only exports.
canExportVideo	Set this to non-zero if the exporter can output video.
canExportAudio	Set this to non-zero if the exporter can output audio.
singleFrameOnly	Set this to non-zero if the exporter makes single frames (used by still image exporters).
maxAudiences	
interfaceVersion	Exporter API version that the plug-in supports.
isCacheable	New in CS5. Set this non-zero to have Premiere Pro cache this exporter.

exExporterInstanceRec

Selector: *exSelBeginInstance* and *exSelEndInstance*

Provides access to the privateData for the indicated filetype, so that the exporter can allocate privateData and pass it to the host, or deallocate it.

```
typedef struct {
     csSDK_uint32   exporterPluginID;
     csSDK_uint32   fileType;
     void*          privateData;
```

```
} exExporterInstanceRec;
```

exporterPluginID	The host's internal identifier for this exporter. Do not modify.
fileType	The file format four character code set by the exporter during *exSelStartup*.
privateData	Data allocated and managed by the exporter.

exGenerateDefaultParamRec

Selector: *exSelGenerateDefaultParams*

Provides access to the `privateData` for the indicated `filetype`, so that the exporter can generate the default parameter set.

```
typedef struct {
     csSDK_uint32   exporterPluginID;
     void*          privateData;
     csSDK_uint32   fileType;
} exExporterInstanceRec;
```

exporterPluginID	The host's internal identifier for this exporter. Do not modify.
privateData	Data allocated and managed by the exporter.
fileType	The file format four character code set by the exporter during *exSelStartup*.

exParamButtonRec

Selector: *exSelParamButton*

Provides access to the `privateData` for the indicated `filetype`, and discloses the specific button hit by the user, since there can be multiple button parameters.

```
typedef struct {
     csSDK_uint32        exporterPluginID;
     void*               privateData;
     csSDK_uint32        fileType;
     csSDK_int32         exportAudio;
     csSDK_int32         exportVideo;
     csSDK_int32         multiGroupIndex;
     exParamIdentifier   buttonParamIdentifier;
} exParamButtonRec;
```

exporterPluginID	The host's internal identifier for this exporter. Do not modify.
privateData	Data allocated and managed by the exporter.
fileType	The file format four character code set by the exporter during *exSelStartup*.
exportAudio	If non-zero, the current settings are set to export audio.
exportVideo	If non-zero, the current settings are set to export video.
multiGroupIndex	Discloses the index of the multi-group, containing the button hit by the user.
buttonParamIdentifier	Discloses the parameter ID of the button hit by the user.

exParamChangedRec

Selector: *exSelValidateParamChanged*

Provides access to the `privateData` for the indicated `filetype`, and discloses the specific parameter changed by the user.

```
typedef struct {
     csSDK_uint32         exporterPluginID;
     void*                privateData;
     csSDK_uint32         fileType;
     csSDK_int32          exportAudio;
     csSDK_int32          exportVideo;
     csSDK_int32          multiGroupIndex;
     exParamIdentifier    changedParamIdentifier;
     csSDK_int32          rebuildAllParams;
} exParamChangedRec;
```

exporterPluginID	The host's internal identifier for this exporter. Do not modify.
privateData	Data allocated and managed by the exporter.
fileType	The file format four character code set by the exporter during *exSelStartup*.
exportAudio	If non-zero, the current settings are set to export audio.
exportVideo	If non-zero, the current settings are set to export video.
multiGroupIndex	Discloses the index of the multi-group, containing the parameter changed by the user.
changedParamIdentifier	Discloses the parameter ID of the parameter changed by the user. May be empty if the changed item was `exportAudio`, `exportVideo` or the current `multiGroupIndex`.
rebuildAllParams	Set this to non-zero to tell the host to reload ALL parameters from the parameter list.

exParamSummaryRec

Selector: *exSelGetParamSummary*

Provides access to the `privateData` for the indicated `filetype`, and provides buffers for the exporter to fill in with a localized summary of the parameters.

```
typedef struct {
     csSDK_uint32   exporterPluginID;
     void*          privateData;
     csSDK_int32    exportAudio;
     csSDK_int32    exportVideo;
     prUTF16Char    Summary1[256];
     prUTF16Char    Summary2[256];
     prUTF16Char    Summary3[256];
} exParamSummaryRec;
```

`exporterPluginID`	The host's internal identifier for this exporter. Do not modify.
`privateData`	Data allocated and managed by the exporter.
`exportAudio`	If non-zero, the current settings are set to export audio.
`exportVideo`	If non-zero, the current settings are set to export video.
`Summary1`	Fill these in with a line of a localized summary of the parameters.
`Summary2`	
`Summary3`	

exPostProcessParamsRec

Selector: *exSelPostProcessParams*

Provides access to the `privateData` for the indicated `filetype`.

```
typedef struct {
     csSDK_uint32        exporterPluginID;
     void*               privateData;
     csSDK_uint32        fileType;
     csSDK_int32         exportAudio;
     csSDK_int32         exportVideo;
} exPostProcessParamsRec;
```

`exporterPluginID`	The host's internal identifier for this exporter. Do not modify.
`privateData`	Data allocated and managed by the exporter.

fileType	The file format four character code set by the exporter during *exSelStartup*.
exportAudio	If non-zero, the current settings are set to export audio.
exportVideo	If non-zero, the current settings are set to export video.

exQueryExportFileExtensionRec

Selector: *exSelQueryExportFileExtension*

Provides access to the privateData for the indicated filetype, and provides a buffer for the exporter to fill in with the file extension.

```
typedef struct {
     csSDK_uint32        exporterPluginID;
     void*               privateData;
     csSDK_uint32        fileType;
     prUTF16Char         outFileExtension[256];
} exQueryExportFileExtensionRec;
```

exporterPluginID	The host's internal identifier for this exporter. Do not modify.
privateData	Data allocated and managed by the exporter.
fileType	The file format four character code set by the exporter during *exSelStartup*.
outFileExtension	Provide the file extension here, given the current parameter settings.

exQueryOutputFileListRec

Selector: *exSelQueryOutputFileList*

Provides access to the privateData for the indicated filetype, and provides a pointer to a array of exOutputFileRecs for the exporter to fill in with the file paths.

```
typedef struct {
     csSDK_uint32        exporterPluginID;
     void*               privateData;
     csSDK_uint32        fileType;
     csSDK_int32         numOutputFiles;
     PrSDKString         path;
     exOutputFileRec     *outputFileRecs;
} exQueryOutputFileListRec;
```

exporterPluginID	The host's internal identifier for this exporter. Do not modify.
privateData	Data allocated and managed by the exporter.
fileType	The file format four character code set by the exporter during *exSelStartup*.
numOutputFiles	On the first call to *exSelQueryOutputFileList*, provide the number of file paths here.
path	New in CS5. Contains the primary intended destination path provided by the host.
outputFileRecs	An array of exOutputFileRecs. On the second call to *exSelQueryOutputFileList*, the path length (including trailing null) for each path. On the third call, fill in the path of each exOutputFileRec. typedef struct { int pathLength; prUTF16Char* path; } exOutputFileRec;

exQueryOutputSettingsRec

Selector: *exSelQueryOutputSettings*

Provides access to the privateData for the indicated filetype, and provides a set of members for the exporter to fill in with the current export settings.

```
typedef struct {
     csSDK_uint32         exporterPluginID;
     void*                privateData;
     csSDK_uint32         fileType;
     csSDK_int32          inMultiGroupIndex;
     csSDK_int32          inExportVideo;
     csSDK_int32          inExportAudio;
     csSDK_int32          outVideoWidth;
     csSDK_int32          outVideoHeight;
     PrTime               outVideoFrameRate;
     csSDK_int32          outVideoAspectNum;
     csSDK_int32          outVideoAspectDen;
     csSDK_int32          outVideoFieldType;
     double               outAudioSampleRate;
     PrAudioSampleType    outAudioSampleType;
     PrAudioChannelType   outAudioChannelType;
     csSDK_uint32         outBitratePerSecond;
```

```
} exQueryOutputSettingsRec;
```

exporterPluginID	The host's internal identifier for this exporter. Do not modify.
privateData	Data allocated and managed by the exporter.
fileType	The file format four character code set by the exporter during *exSelStartup*.
inMultiGroupIndex	Return the parameter settings of the multi-group with this index.
inExportVideo	If non-zero, the current settings are set to export video.
inExportAudio	If non-zero, the current settings are set to export audio.
outVideoWidth outVideoHeight ...	Return each parameter setting, by getting the current value of the parameter using the Export Param Suite. Some settings, such as outVideoFieldType, may be implicit, for example if the format only supports progressive frames.

exQueryStillSequenceRec

Selector: *exSelQueryStillSequence*

Provides access to the privateData for the indicated filetype, and provides a set of members for the exporter to provide information on how it would export the sequence of stills.

```
typedef struct {
     csSDK_uint32   exporterPluginID;
     void*          privateData;
     csSDK_uint32   fileType;
     csSDK_int32    exportAsStillSequence;
     PrTime         exportFrameRate;
} exQueryStillSequenceRec;
```

exporterPluginID	The host's internal identifier for this exporter. Do not modify.
privateData	Data allocated and managed by the exporter.
fileType	The file format four character code set by the exporter during *exSelStartup*.
exportAsStillSequence	Set this to non-zero to tell the host that the exporter can export the stills as a sequence.
exportFrameRate	Set this to the frame rate of the still sequence.

exValidateOutputSettingsRec

Selector: *exSelValidateOutputSettings*

Provides access to the `privateData` for the indicated `filetype`, so that the exporter can validate the current parameter settings.

```
typedef struct {
     csSDK_uint32   exporterPluginID;
     void*          privateData;
     csSDK_uint32   fileType;
} exExporterInstanceRec;
```

`exporterPluginID`	The host's internal identifier for this exporter. Do not modify.
`privateData`	Data allocated and managed by the exporter.
`fileType`	The file format four character code set by the exporter during *exSelStartup*.

Suites

For information on how to acquire and manage suites, see the SweetPea Suites section.

Export File Suite

A cross-platform suite for writing to files on disk. Also provides a call to get the file path, given the file object. See PrSDKExportFileSuite.h.

Export Info Suite

GetExportSourceInfo

Get information on the source currently being exported.

```
prSuiteError (*GetExportSourceInfo)(
     csSDK_uint32                inExporterPluginID,
     PrExportSourceInfoSelector  inSelector,
     PrParam                     *outSourceInfo);
```

Value	Type	Description
`kExportInfo_VideoWidth`	`Int32`	Width of source video
`kExportInfo_VideoHeight`	`Int32`	Height of source video
`kExportInfo_VideoFrameRate`	`PrTime`	Frame rate

kExportInfo_VideoFieldType	Int32	One of the prFieldType values
kExportInfo_VideoDuration	Int64	A PrTime value
kExportInfo_ PixelAspectNumerator	Int32	Pixel aspect ratio (PAR) numerator
kExportInfo_ PixelAspectDenominator	Int32	Pixel aspect ratio denominator
kExportInfo_AudioDuration	Int64	A PrTime value
kExportInfo_ AudioChannelsType	Int32	One of the PrAudioChannelType values. Returns 0 (which is undefined) if there's no audio.
kExportInfo_AudioSampleRate	Float64	
kExportInfo_ProjectPath kExportInfo_ProjectLinkAtom kExportInfo_CreatorAtom		Unimplemented in CS4.
kExportInfo_SourceHasAudio	Bool	Non-zero if source has audio
kExportInfo_SourceHasVideo	Bool	Non-zero if source has video
kExportInfo_RenderAsPreview	Bool	Returns a non-zero value if currently rendering preview files.
kExportInfo_SequenceGUID	Guid	A PrPluginID, which is a unique GUID for the sequence.
kExportInfo_SessionFilePath	PrMemoryPtr	A prUTF16Char array. The exporter should release the pointer using the Memory Manager suite.

Export Param Suite

Specify all parameters for your exporter UI. See PrSDKExportParamSuite.h. Also, see the SDK Export sample for a demonstration of how to use this suite.

As of CS5, there is an issue where width and height ranges aren't correctly set (bug 2570438). You may notice this when adjusting the width and height in the Export Settings UI. By unclicking the chain that constrains width and height ratio, you will be able to modify the width and height. As a side-effect of this bug, if the exporter is used to render preview files in an Editing Mode, the user will be able to choose any preview frame size between 24x24 and 10240x8192.

Export Progress Suite

Report progress during the export. Also, handle the case where the user pauses an export. See PrSDKExportProgressSuite.h.

Palette Suite

A seldom-used suite for palettizing an image, for example, for GIFs. See PrSDKPaletteSuite.h.

Sequence Audio Suite

Get audio from the host.

MakeAudioRenderer

Create an audio renderer, in preparation to get rendered audio from the host.

```
prSuiteError (*MakeAudioRenderer)(
     csSDK_uint32         inPluginID,
     PrTime               inStartTime,
     PrAudioChannelType   inChannelType,
     PrAudioSampleType    inSampleType,
     float                inSampleRate,
     csSDK_uint32*        outAudioRenderID);
```

Parameter	Description
inPluginID	Pass in exporterPluginID from exDoExportRec.
inStartTime	Start time for the audio requests.
inChannelType	PrAudioChannelType enum value for the channel type needed.
inSampleType	PrAudioSampleType enum value for the sample type needed.
inSampleRate	Samples per second.
outAudioRenderID	This ID passed back is needed for subsequent calls to this suite.

ReleaseAudioRenderer

Release the audio renderer when the exporter is done requesting audio.

```
prSuiteError (*ReleaseAudioRenderer)(
     csSDK_uint32    inPluginID,
     csSDK_uint32    inAudioRenderID);
```

Parameter	Description
inPluginID	Pass in exporterPluginID from exDoExportRec.
inAudioRenderID	The call will release the audio renderer with this ID.

GetAudio

Returns from the host the next contiguous requested number of audio sample frames, specified in inFrameCount, in inBuffer as arrays of uninterleaved floating point values. Returns suiteError_NoError if no error. The plug-in must manage the memory allocation of inBuffer, which must point to n buffers of floating point values of length inFrameCount, where n is the number of channels. When inClipAudio is non-zero, this parameter makes GetAudio clip the audio samples at +/- 1.0.

```
prSuiteError (*GetAudio)(
     csSDK_uint32    inAudioRenderID,
     csSDK_uint32    inFrameCount,
     float**         inBuffer,
     char            inClipAudio);
```

Parameter	Description
inAudioRenderID	Pass in the outAudioRenderID returned from MakeAudioRenderer(). This gives the host the context of the audio render.
inFrameCount	The number of audio frames to return in inBuffer. The next contiguous audio frames will always be returned, unless ResetAudioToBeginning has just been called.
inBuffer	An array of float arrays, allocated by the exporter. The host returns the samples for each audio channel in a separate array.
inClipAudio	When true, GetAudio will return audio clipped at +/- 1.0. Otherwise, it will return unclipped audio.

ResetAudioToBeginning

This call will reset the position on the audio generation to time zero. This can be used for multi-pass encoding.

```
prSuiteError (*ResetAudioToBeginning)(
     csSDK_uint32 inAudioRenderID);
```

GetMaxBlip

Returns the maximum number of audio sample frames that can be requested from one call to `GetAudio` in `maxBlipSize`.

```
prSuiteError (*GetMaxBlip)(
     csSDK_uint32   inAudioRenderID,
     PrTime    inTicksPerFrame,
     csSDK_uint32*  maxBlipSize);
```

Sequence Render Suite

Get rendered video from one of the renderers available to the host. This may use the host's built-in renderer, or a plug-in renderer, if available For best performance, use the asynchronous render requests with the source media prefetching calls, although synchronous rendering is available too.

MakeVideoRenderer()

Create a video renderer, in preparation to get rendered video.

```
prSuiteError (*MakeVideoRenderer)(
     csSDK_uint32   pluginID,
     csSDK_uint32*  outVideoRenderID
     PrTime         inFrameRate);
```

Parameter	Description
`pluginID`	Pass in `exporterPluginID` from `exDoExportRec`.
`outVideoRenderID`	This ID passed back is needed for subsequent calls to this suite.
`inFrameRate`	Frame rate, in ticks.

ReleaseVideoRenderer()

Release the video renderer when the exporter is done requesting video.

```
prSuiteError (*ReleaseVideoRenderer)(
     csSDK_uint32   pluginID,
     csSDK_uint32   inVideoRenderID);
```

Parameter	Description
pluginID	Pass in exporterPluginID from exDoExportRec.
inVideoRenderID	The call will release the video renderer with this ID.

struct SequenceRender_ParamsRec

Fill this structure in before calling RenderVideoFrame(), QueueAsyncVideoFrameRender(), or PrefetchMediaWithRenderParameters(). Note that if the frame aspect ratio of the request does not match that of the sequence, the frame will be letterboxed or pillarboxed, rather than stretched to fit the frame.

```
typedef struct
{
     const PrPixelFormat*     inRequestedPixelFormatArray;
     csSDK_int32              inRequestedPixelFormatArrayCount;
     csSDK_int32              inWidth;
     csSDK_int32              inHeight;
     csSDK_int32              inPixelAspectRatioNumerator;
     csSDK_int32              inPixelAspectRatioDenominator;
     PrRenderQuality          inRenderQuality;
     prFieldType              inFieldType;
     csSDK_int32              inDeinterlace;
     PrRenderQuality          inDeinterlaceQuality;
     csSDK_int32              inCompositeOnBlack;
} SequenceRender_ParamsRec;
```

Member	Description
inRequestedPixelFormatArray	An array of PrPixelFormats that list your format preferences in order.
inRequestedPixelFormatArrayCount	Size of the pixel format array.
inWidth	Width to render at.
inHeight	Height to render at.
inPixelAspectRatioNumerator	Numerator of the pixel aspect ratio.
inPixelAspectRatioDenominator	Denominator of the pixel aspect ratio.

inRenderQuality	Use one of the PrRenderQuality enumerated values.
inFieldType	Use one of the prFieldType constants.
inDeinterlace	Set to non-zero, to deinterlace.
inDeinterlaceQuality	Use one of the PrRenderQuality enumerated values.
inCompositeOnBlack	Set to non-zero, to composite the render on black.

struct SequenceRender_GetFrameReturnRec

Returned from RenderVideoFrame() and passed by PrSDKSequenceAsyncRenderCompletionProc().

```
typedef struct
{
     void*          asyncCompletionData;
     csSDK_int32    returnVal;
     csSDK_int32    repeatCount;
     csSDK_int32    onMarker;
     PPixHand       outFrame;
} SequenceRender_GetFrameReturnRec;
```

Member	Description
asyncCompletionData	Passed to PrSDKSequenceAsyncRenderCompletionProc() from QueueAsyncVideoFrameRender(). Not used by RenderVideoFrame().
returnVal	ErrNone , Abort, Done, or an error code.
repeatCount	The number of repeated frames from this frame forward. In the output file, this could be writing NULL frames, changing the current frame's duration, or whatever is appropriate according to the codec.
onMarker	If non-zero, there is a marker on this frame.
outFrame	Returned from RenderVideoFrame(). Not returned from PrSDKSequenceAsyncRenderCompletionProc()

RenderVideoFrame()

The basic, synchronous call to get a rendered frame from the host. Returns suiteError_NoError if you can continue exporting, exportReturn_Abort if the user aborted the export, exportReturn_Done if the export has finished, or an error code.

```
prSuiteError (*RenderVideoFrame)(
     csSDK_uint32                          inVideoRenderID,
     PrTime                                inTime,
     SequenceRender_ParamsRec*             inRenderParams,
     PrRenderCacheType                     inCacheFlags,
     SequenceRender_GetFrameReturnRec*     getFrameReturn);
```

Parameter	Description
inVideoRenderID	Pass in the outVideoRenderID returned from MakeVideoRenderer(). This gives the host the context of the video render.
inTime	The frame time requested.
inRenderParams	The details of the render.
inCacheFlags	One or more cache flags.
getFrameReturn	Passes back a structure that contains info about the frame returned, and the rendered frame itself.

GetFrameInfo()

Gets information about a given frame. Currently, SequenceRender_FrameInfoRec only contains repeatCount, which is the number of repeated frames from this frame forward.

```
prSuiteError (*GetFrameInfo)(
     csSDK_uint32                  inVideoRenderID,
     PrTime                        inTime,
     SequenceRender_FrameInfoRec*  outFrameInfo);
```

SetAsyncRenderCompletionProc()

Register a notification callback for getting asynchronously rendered frames when the render completes. asyncGetFrameCallback should have the signature described in PrSDKSequenceAsyncRenderCompletionProc below.

```
prSuiteError (*SetAsyncRenderCompletionProc)(
     csSDK_uint32                          inVideoRenderID,
     PrSDKSequenceAsyncRenderCompletionProc asyncGetFrameCallback,
     long                                  callbackRef);
```

Parameter	Description

inVideoRenderID	Pass in the outVideoRenderID returned from MakeVideoRenderer(). This will be passed to PrSDKSequenceAsyncRenderCompletionProc.
asyncGetFrameCallback	The notification callback.
inCallbackRef	A pointer holding data private to the exporter. This could be, for example, a pointer to an exporter instance. This will also be passed to PrSDKSequenceAsyncRenderCompletionProc.

PrSDKSequenceAsyncRenderCompletionProc()

Use this function signature for your callback used for async frame notification, passed to SetAsyncRenderCompletionProc. Error status (error or abort) is returned in inGetFrameReturn.

```
void (*PrSDKSequenceAsyncRenderCompletionProc)(
     csSDK_uint32                          inVideoRenderID,
     void*                                 inCallbackRef,
     PrTime                                inTime,
     PPixHand                              inRenderedFrame,
     SequenceRender_GetFrameReturnRec      *inGetFrameReturn);
```

Parameter	Description
inVideoRenderID	The outVideoRenderID that the exporter passed to SetAsyncRenderCompletionProc earlier.
inCallbackRef	A pointer that the exporter sets using SetAsyncRenderCompletionProc(). This could be, for example, a pointer to an exporter instance.
inTime	The frame time requested.
inRenderedFrame	The rendered frame. The exporter is reponsible for disposing of this PPixHand using the Dispose() call in the PPix Suite.
inGetFrameReturn	A structure that contains info about the frame returned, and it includes the inAsyncCompletionData originally passed to QueueAsyncVideoFrameRender().

QueueAsyncVideoFrameRender()

Use this call rather than RenderVideoFrame() to queue up a request to render a specific frame asynchronously. The rendering can happen on a separate thread or processor. When the

render is completed, the `PrSDKSequenceAsyncRenderCompletionProc` that was set using `SetAsyncRenderCompletionProc` will be called.

```
prSuiteError (*QueueAsyncVideoFrameRender)(
     csSDK_uint32                    inVideoRenderID,
     PrTime                          inTime,
     csSDK_uint32*                   outRequestID,
     SequenceRender_ParamsRec*       inRenderParams,
     PrRenderCacheType               inCacheFlags,
     void*                           inAsyncCompletionData);
```

Parameter	Description
`inVideoRenderID`	Pass in the `outVideoRenderID` returned from `MakeVideoRenderer()`. This gives the host the context of the video render.
`inTime`	The frame time requested.
`outRequestID`	Passes back a request ID, which... doesn't seem to have any use.
`inRenderParams`	The details of the render.
`inCacheFlags`	One or more cache flags.
`inAsyncCompletionData`	This data will be passed to the `PrSDKSequenceAsyncRenderCompletionProc` in `inGetFrameReturn.asyncCompletionData`.

PrefetchMedia()

Prefetch the media needed to render this frame. This is a hint to the importers to begin reading media needed to render this video frame.

```
prSuiteError (*PrefetchMedia)(
     csSDK_uint32    inVideoRenderID,
     PrTime          inFrame);
```

PrefetchMediaWithRenderParameters()

Prefetch the media needed to render this frame, using all of the parameters used to render the frame. This is a hint to the importers to begin reading media needed to render this video frame.

```
prSuiteError (*PrefetchMediaWithRenderParameters)(
     csSDK_uint32                    inVideoRenderID,
     PrTime                          inTime,
     SequenceRender_ParamsRec*       inRenderParams);
```

CancelAllOutstandingMediaPrefetches()

Cancel all media prefetches that are still outstanding.

```
prSuiteError (*PrefetchMedia)(
     csSDK_uint32   inVideoRenderID);
```

IsPrefetchedMediaReady()

Check on the status of a prefetch request.

```
prSuiteError (*IsPrefetchedMediaReady)(
     csSDK_uint32   inVideoRenderID,
     PrTime         inTime,
     prBool*        outMediaReady);
```

MakeVideoRendererForTimeline()

Similar to `MakeVideoRenderer`, but for use by renderer plug-ins. Creates a video renderer, in preparation to get rendered video from the host. The TimelineID in question must refer to a top-level sequence.

```
prSuiteError (*MakeVideoRendererForTimeline)(
     PrTimelineID   inTimeline,
     csSDK_uint32*  outVideoRendererID);
```

MakeVideoRendererForTimelineWithFrameRate()

Similar to `MakeVideoRendererForTimeline`, with an additional frame rate parameter. This is useful for the case of a nested multicam sequence.

```
prSuiteError (*MakeVideoRendererForTimelineWithFrameRate)(
     PrTimelineID   inTimeline,
     PrTime         inFrameRate,
     csSDK_uint32*  outVideoRendererID);
```

ReleaseVideoRendererForTimeline()

Similar to `ReleaseVideoRenderer`, but for use by renderer plug-ins. Release the video renderer when the renderer plug-in is done requesting video.

```
prSuiteError (*ReleaseVideoRendererForTimeline)(
     csSDK_uint32   inVideoRendererID);
```

Additional Details

Multiplexer Tab Ordering

If your exporter provides a Multiplexer tab like some of ours do, you will find that it appears after the Video and Audio tab, rather than before those tabs as in the case of our exporters. The key is to use the following define as the parameter identifer for the multiplexer tab group:
`#define ADBEMultiplexerTabGroup "ADBEAudienceTabGroup"`

Creating a Non-Editable String in the Parameter UI

During *exSelGenerateDefaultParams*, add a parameter with `exNewParamInfo.flags = exParamFlag_none`. Then during *exSelPostProcessParams*, call `AddConstrainedValuePair()` in the Export Param Suite. f you only add one value pair, then the parameter will be a non-editable string. In the case of the SDK Exporter sample, it adds two, which appear as a drop-down option.

Accelerated Renderers

Accelerated renderers are a new plug-in type in CS4, defined in PrSDKAcceleratedRender.h. At the top-level, a renderer statelessly advertises a list of pixel formats that it supports, including custom formats, but also including standard formats (allowing the acceleration of renders for built-in exporters).

Since 3rd-party renderers need to take over rendering for certain segments of a sequence, there is a priority mechanism similar to the importer mechanism. Plug-ins can be prioritized above or below the built-in renderer.

Given a specific sequence, the host will setup a state object for rendering that sequence using a specific renderer. The renderer will query the sequence using the same suite functions that the player currently uses to do the same. This state will include a callback function that will act much like the real-time status querying that is done in the player, allowing a specific renderer to accelerate only certain ranges of a sequence.

For a given export, this initialization will be performed for each loaded renderer. At the end of the export each will be torn down.

During the export, for a given frame request the host will choose the highest priority renderer that supports that frame, for a requested pixel format. A renderer at a lower priority than the built-in renderer can be selected if a custom pixel format is requested.

The renderer interface will be completely asynchronous. The request can be initiated, queried for completion and canceled. The request will include a callback function which will be called if there is an error, if the render completes, or if the host cancels it. There should be no instance where the request is initiated without error, and the callback is not called. The request will include a RequestID object, and a suite will allow the renderer to query this for properties (pixel formats, deinterlacing options, etc.).

The renderer will allocate PPixs using the PPix Creator Suite. If the renderer produces a frame in a custom format, it will pass through to the exporter, as above. The host will do no post processing, and the renderer must support the complete frame render (or not at all). The host will, in the case of exporting through Media Encoder need to copy the frame to another process, so again, the frame data must be flat.

Effects and transition plugins will not support the custom pixel formats. To enable accelerated effects and transitions, this must be done at the renderer level. The renderer can access importers using the Clip Render Suite.

These plug-ins are loaded, and initialized in PProHeadless. When a Media Encoder export renders through PProHeadless, an accelerated renderer is created for the sequence, and queried for the segments and pixel formats that it can handle.

In PProHeadless, when a renderer is available for a given render request, it will be used. Render requests will be made and tracked, and can be cancelled by the dynamic link client. Multiple renderer plugins can be loaded, and will be queried in priority order.

DynamicLinkClient now supports requests for custom pixel formats, and will forward that on to DynamicLinkServer, which will use the custom formats when deciding on a renderer. At the moment, all custom formats are assumed to be preferred over the selected native 4444 format.

Guidelines for Exporters in Encore

New in CS5, third-party exporters can now be used to transcode assets to MPEG-2 or Blu-ray compliant files. Currently, the option to choose a third-party exporter is only available on a per-clip basis, not on a project-wide basis. The user will need to right-click on an asset in the Project panel, choose Transcode Settings, and choose the third-party preset from the Quality Preset drop-down.

Encore will remain a 32-bit application for CS5. So if you are developing plug-ins for Encore, use the CS5 headers to create 32-bit plug-ins. We have left the 32-bit configurations in the

sample projects to facilitate this. Install the exporter in the Encore application folder at Plug-ins/Common/. Note that on Mac OS, this subfolder is in within the application package.

Naming Your Exporter

Encore only uses the MPEG2-DVD and MPEG2 Blu-ray formats for transcoding to MPEG2-DVD and MPEG2 Blu-ray formats, respectively. Currently it looks for the substrings "MPEG2-DVD", "MPEG2 Blu-ray" and "H.264 Blu-ray" in the exporter name to identify the video format of the exporter, and to enable it within the Encore UI. So the format name returned from exporter plug-in should contain one of these as a substring, in order for it to be usable within Encore. For example "My MPEG2 Blu-ray", "Accelerated MPEG2-DVD", etc. Please avoid using the exact same names as the built-in formats to avoid conflict.

Naming Your Output

Encore uses the exporters to create elementary video and audio streams (muxing is switched to off during transcoding). The output file extensions should be standard ones: .m2v for MPEG2-DVD, MPEG2 Blu-ray video formats, .m4v for H.264 Blu-ray; .ac3 for Dolby audio, .wav for PCM, .mpa for MPEG-1 Layer 2.

Parameters

Please refer to the built-in MPEG2-DVD and MPEG2 Blu-ray formats present in Encore to get familiar with the typical exporter user interface in Encore. Having UI properties similar to the built-in formats in Encore will make it easier to integrate a third-party exporter.

The audio formats available in an exporter should correspond to the same choices as available in Encore for a DVD or Blu-ray project. In an MPEG-2 DVD exporter, the audio formats should be either Dolby Digital 2.0 (stereo), MPEG-1 Layer 2 audio in stereo or PCM audio (48kHz). For an MPEG2 Blu-ray exporter, only the Dolby and the PCM formats should be available. For more details regarding audio formats supported in Encore, please refer to the Encore help documentation. Allowing audio formats other than these for encoding will not work in Encore due to the constraints of the DVD/Blu-ray disc specifications.

Encore will need to access many of the exporter's encoding parameters. It may even modify some of the encoding parameters during the transcoding to MPEG-2 DVD and Blu-ray formats, so that the encoding stays within the bit-budget constraints of the project. So a third-party exporter must use specific property identifiers and property types. If these parameters are not used, then there is little guarantee of the correctness of the encoded file and the size of the final disc, since Encore will not be able to control the settings of the exporter to apply the size constraints to the output files. Below is a list of the properties with their identifiers and types that an exporter plugin must support:

Property Identifier	Property type	Description
ADBEVideoWidth (required)	exParamType_int	Frame width
ADBEVideoHeight (required)	exParamType_int	Frame height
ADBEVideoVBR	exParamType_int Constrained value list	Type of encoding (constant/variable bitrate, 1 / 2 passes) 0 = CBR 1 = VBR, 1 Pass 2 = VBR, 2 Pass
ADBEVideoBitRate ADBEVideoMaxBitRate ADBEVideoAvgBitRate ADBEVideoMinBitRate	exParamType_float	Video bitrate(s) (Mbps) For CBR encoding use the first parameter. For VBR encoding use parameters 2-4.
ADBEVideoFPS (required)	exParamType_ ticksFrameRate	Frame rate
ADBEMPEGCodec-BroadcastStandard (required)	exParamType_int Constrained value list	0 = NTSC 1 = PAL 2 = SECAM
ADBEVideoAspect	exParamType_int Constrained value list	Frame aspect ratio 1 = Square 1:1 2 = Standard 4:3 3 = Widescreen16:9
ADBEVMCMux_Type	exParamType_int Constrained value list	Encore needs a way to switch off muxing as it creates only elementary streams 0 = MPEG-1 1 = VCD 2 = MPEG-2 3 = SVCD 4 = DVD 5 = TS 6 = None
ADBEVideoFieldType	exParamType_int Constrained value list	0 = Progressive 1 = Upper field first 2 = Lower field first
ADBEAudioCodec (required)	exParamType_int Constrained value list	Use these 4CCs for values 'dlby' – Dolby 'PCMA' – PCM 'mpa ' – MPEG-1 Layer 2

ADBEAudio_Endianness (optional)	exParamType_int Constrained value list	If using Dolby audio; Encore will set to big endian for AC3 files 0 = little endian 1 = big endian
ADBEAudioBitrate (required for Dolby and MPEG-2 audio codecs)	exParamType_int	Audio codec bitrate (kbps)

Guidelines for Exporters in Premiere Elements

First, make sure you are building the exporter using the right SDK. Premiere Elements 8 requires the Premiere Pro CS4 SDK. The next version of Premiere Elements will likely use the CS5 SDK.

Exporter Preset

For an exporter to show up in the Premiere Elements UI, you'll need to install a preset in a specific location:

1) Create a folder named "OTHERS" in [App installation folder]/sharingcenter/Presets/pc/
2) Create a sub-folder with your name (e.g. MyCompany) under OTHERS and place the preset file (.epr) in it.
The final path of the preset file should be something like [App installation folder]/sharingcenter/Presets/pc/OTHERS/MyCompany/MyPreset.epr
3) Relaunch Premiere Elements.
    a. Add a clip to the timeline
    b. Goto the "Share" tab
    c. Under that choose "Personal Computer"
    d. You should see the "Others – 3rd Party Plug-ins" in the list of formats. Select this.
    e. Your preset should be seen in the drop-down.

Return Values

Premiere Elements 8 uses a slightly different definition of the return values. Use the following definition instead:

```
enum
{
      exportReturn_ErrNone = 0,
      exportReturn_Abort,
      exportReturn_Done,
      exportReturn_InternalError,
```

```
        exportReturn_OutputFormatAccept,
        exportReturn_OutputFormatDecline,
        exportReturn_OutOfDiskSpace,
        exportReturn_BufferFull,
        exportReturn_ErrOther,
        exportReturn_ErrMemory,
        exportReturn_ErrFileNotFound,
        exportReturn_ErrTooManyOpenFiles,
        exportReturn_ErrPermErr,
        exportReturn_ErrOpenErr,
        exportReturn_ErrInvalidDrive,
        exportReturn_ErrDupFile,
        exportReturn_ErrIo,
        exportReturn_ErrInUse,
        exportReturn_IterateExporter,
        exportReturn_IterateExporterDone,
        exportReturn_InternalErrorSilent,
        exportReturn_ErrCodecBadInput,
        exportReturn_ErrLastErrorSet,
        exportReturn_ErrLastWarningSet,
        exportReturn_ErrLastInfoSet,
        exportReturn_ErrExceedsMaxFormatDuration,
        exportReturn_VideoCodecNeedsActivation,
        exportReturn_AudioCodecNeedsActivation,
        exportReturn_IncompatibleAudioChannelType,
        exportReturn_Unsupported = -100
};
```

The red values are unique to Premiere Elements 8, and shifted the subsequent return values 2 values higher than their definition in the Premiere Pro SDK.

8 Players

Players manage playback of video in the Source Monitor, Sequence Monitor, Multicam Monitor, or Reference Monitor. Players can also play video out to a hardware device such as an SDI card or other third-party hardware. Only one player per sequence can be used for playback, and once the player is set, it cannot be changed. So a player should be designed to handle playback from the start of the editing process all the way to final playout. The player used by the sequence is determined while creating a new sequence, by the Editing Mode selected. An Editing Mode is a pairing of a player and a exporter. The player used for playback of a single clip in the Source Monitor is determined by the player selected in Preferences > Player Settings > Default Player. So this player may not be the same as the player selected in the Editing Mode.

A player can call the host to render frames using the Playmod Render Suite, or it can take over rendering for any segment it chooses. A player may choose to take over rendering if it can render certain effects or composite faster than the host's renderer. However, a player may not wish to take over rendering every segment of a sequence, since there may be certain effects or file formats that it cannot render.

There may be several instances of a player at a time, each with a unique `playID`. One instance is created when a project is opened for the titler, for external monitor output. Another instance is created for each sequence that has been opened in the Sequence Monitor. Another instance may be created when a single clip is opened in the Source Monitor. Another instance may be for a Reference Monitor to the Sequence Monitor. Yet another instance may be for a transition preview in the Effect Controls Panel (only certain transitions have a transition preview). Only one of these players will be active at a time, and each player will receive messages from the host to activate and deactivate. A player will receive messages to playback or scrub only when it is active.

All sequence players must manage their own private representation of the sequence called a cut-list -- a list of tree structures of video clips and effects. The host sends messages to the appropriate player instance as the user makes edits to the sequence, so that the player can use the Video Segment Suite to find out exactly what parts of the sequence have changed, and make corresponding updates to its cutlist. Certain edits may result in cutlist updates to several player instances, for example, if the user makes changes to a sequence that is nested within another sequence.

If you've never developed a player before, you can skip the What's New sections, and go directly to Getting Started.

What's New

What's New in Premiere Pro CS5?

New settings in the Monitor panel give the user control over which field to display, and whether rendering for the display should be done at full resolution or at a lower resolution. To provide API access to these new settings, a new selector has been added: *playmod_SetDisplayStateProperties*. Players should set `PMCapSupportsDisplayStateProperties` in `pmModuleInfoRec.capabilityFlags` to receive this new selector.

New in CS5, `GetAcceleratedRendererRTStatusForTime` in the Playmod Render Suite provides a way for a player to find out if a segment uses effects accelerated by the Mercury Playback Engine or any other accelerated renderer.

Sequences can now have video preview resolution settings that differ from the resolution of the sequence itself. This preview resolution is not used as the resolution sent with *playmod_NewList*, but is used as the resolution preview files are generated at. Since the preview resolution can be changed on-the-fly, this new setting can be useful for working at proxy sizes.

The new optional *playmod_PutTemporaryTimeline* selector brings back the old *playmod_PutSegment* functionality, allowing a player to provide its own rendering of a frame to be displayed on external hardware during an edit.

Returning `playmod_ErrNone` from *playmod_PutFrame* will no longer serve as a signal to the host to draw the frame in the Monitor panel, as was the case previously on Windows. Any player that implements *playmod_PutFrame* must draw the frame in the Monitor panel.

What's New in Premiere Pro CS4?

New Timeline Segments

The new video segments replace the old segments as the way to get information about sequences in the timeline. The player is notified of a change to a sequence during *playmod_VideoSequenceHasChanged*. There are no segments sent with this selector. Rather than the host sending the video segments to the player, the player must request specific details using the Video Segment Suite.

The structure of segments has changed. Previously for segments with multiple layers, there was a layer segment with two inputs, one of which was a layer segment with two inputs, one of which was a layer segment with two inputs, and so on. Now there is a Compositor node with n inputs. Each of those inputs is a Clip node, which has one input which is a Media node, and it also has n Operators, which are effects.

So, a simple example, three clips in a stack, the top one with three effects looked like this before:

```
Segment
     Clip
          Layer
               Clip (foreground layer)
                    Effect (Motion, with keyframes)
                         Effect (Color Corrector)
                              Effect (Blur)
                                   File (top clip)
               Clip (background layer)
                    Layer
                         Clip (foreground layer)
                              File (middle clip)
                         Clip (background layer)
                              File (middle clip)
```

Now it looks like:

```
Segment
     Compositor Node
          Clip Node
               Media Node (bottom clip)
          Clip Node
               Media Node (middle clip)
          Clip Node
               Media Node (top clip)
               Clip Operators (Blur, Color Corrector, Motion)
```

To get a good idea of the segment structure, try the SDK player on Windows, create a sequence using the SDK Editing Mode, and watch the text overlay in the Sequence Monitor as you perform edits.

Reporting Real-Time Status

Providing RT status is fairly similar to before. Respond to the new *playmod_GetRTStatusForTime* selector by filling in the two last members of `prtPlayableRangeRec`.

prtPlayableRangeRec has been changed. The player must now provide not only the real-time status as a PRT_PLAYCODE, but also the outEndTime at the end of the range starting at inStartTime where the real-time status is constant.

Many PRT_PLAYCODEs have been removed. Premiere hadn't been treating these any differently than PRT_PLAYCODE_NON_REALTIME_UNSPECIFIED.

invalRTRanges() has been removed, but an equivalent call is available in the Playmod Render Suite, called RefreshRTStatus.

Sequence-Specific Settings

Sequences in the same project can now have different settings. So there is no longer a single editing mode per project. This means that the Editing Mode suite and most selectors for the getSettings callback in the utilFuncs are no longer supported.

Fractional Resolution

New in Premiere Pro 4.1. Players can optionally support fractional resolution display. Rather than decoding and rendering video at full resolution, it can use a fractional resolution. Fractional decoding is dependent on the importer supporting import at fractional resolution, providing fractional sizes for *imGetPreferredFrameSize*. Premiere Pro 4.1 currently only uses fractional resolution for the Red editing modes, so you can use those editing modes to experiment with the feature.

There are various ways for the user to reach the fractional resolution modes in the UI: From the monitor output menu, monitor wingtip, context menu in the grey area of a monitor background, and the context menu over the video in the window. When in fractional resolution mode, the fractional resolution menu items replace the normal High/Draft/Auto Quality modes.

The player must set PMCapCanDoFractionalResolution during *playmod_GetIndFormat*, which will turn on fractional resolution capabilities for all editing modes supported by the player. The player will get two new selectors, *playmod_SetUseFractionalResolution* and *playmod_SetFractionalResolution*, to notify the player which fractional resolution mode to use.

New RT status

Players can now mark a segment yellow, so that it is not rendered when previewing the work area, but is rendered before export to tape. Use the new prtPlaycode, PRT_PLAYCODE_REALTIME_WITH_MISMATCH.

Other Changes

Players no longer receive the quieting calls *playmod_ActivateFile8* and *playmod_ActivateFile*, or *playmod_PutSegment*.

What's New in Premiere Pro CS3?

A new clip `type` in `prtClipRec`, `PRT_CLIPTYPE_HOSTRENDER`, tells the player that the clip must be rendered by the host because it has variable speed.

All player instances are now sequence players, which means *playmod_NewList* is now always sent rather than *playmod_Open* for players in the Source Monitor. The player for the current editing mode is called for all types of clips in the Source Monitor, not just ones it specifies in *playmod_GetIndFormat*. This enables all types of clips to be previewed on the external monitor, and the audio to be played out through the device used by the player.

A player without a cutlist or window handle is created when a clip is selected in the timeline, and the Effect Controls panel is open. This new player is used when the user hits the audio preview button, to play out audio to the device used by the player. When the clip is deselected or deleted, the player is closed.

New enum value in `PrVideoDisplayType`: `kPrVideoDisplayType_AudioOnly`. This is the display type sent with *playmod_SetVideoDisplayType* when playing an audio-only clip in the Source Monitor. This tells the player instance to turn off video display, and to not round the position returned in *playmod_GetPos* to video frame boundaries. This allows for smoother current time indicator movement, and prevents the CTI from jumping back to the previous video frame boundary when playback begins. `pmPlayerDisplay_AudioOnly` was also added to `pmDisplayMode`, for players still using the legacy *playmod_SetDisplayMode* selector.

The RenderScope call in the Scope Render Suite now renders to a `prWnd` rather than a `prOffscreen`. So the Scope Render Suite version has been incremented to 2.

Field type, audio sample rate, and video subtype are passed in `pmNewListParms`.

New Stock Image Suite, which provides four images useful for players: color bars, not yet rendered, playing on hardware, and capture preview on hardware.

Note: `showFileFrame` only works in Windows.

Clip Render Suite is now at version 2, adding a new call, `GetClipFieldType`.

Getting Started

Selector Calling Sequence

playmod_Startup is sent once at application start-up. *playmod_GetIndFormat* is sent repeatedly so the player can enumerate filetypes and subtypes it plays. This is how the host determines which player is used for which editing mode.

When a new sequence is created that uses the player as part of the sequence's editing mode, *playmod_NewList* is sent, telling the player to create a new cutlist. This selector is also sent to the player specified by Preferences > Player Settings > Default Player, each time a clip is opened in the Source Monitor. *playmod_GetInfo* is then sent to retrieve additional playback information. *playmod_VideoSequenceHasChanged* is then sent, signalling the start of the cutlist updates. Since the project has just been created, the cutlist is empty. As usual after a cutlist update, the host sends *playmod_GetRTStatusForTime* to query the player for the real-time status of each segment in the cutlist. *playmod_Activate* is then sent to tell the player it is the active player, and can output video to the screen or external device. Next, *playmod_SetDisp* tells the player the coordinates on the screen where it should draw. *playmod_SetQuality* is also sent to give the player the initial playback quality setting, modifiable by the user. Finally, *playmod_Update* is sent to tell the player to output the video at the current time to the screen or external device. Then, *playmod_SetPos* tells the player what the current time is. *playmod_GetPos* may be called over and over again, to make sure the current time position has not changed.

If a clip is modified in a sequence, the host sends messages to the sequence player so that it can update its private representation of the cutlist. These messages include *playmod_VideoSequenceHasChanged*, *playmod_GetRTStatusForTime*, and *playmod_Update*. There may be several *playmod_GetRTStatusForTime* calls, if the clip spans multiple segments. If a segment is changed, one will be added in its place; if a segment is removed, an empty one will be added in its place.

A player will get many *playmod_Update* messages whenever the display area is obscured and uncovered. *playmod_Activate* will be sent to activate or deactivate an instance of a player when switching back and forth between player windows (e.g. Sequence Monitor and Reference Monitor), or when the application gains or loses focus.

When the user scrubs in a clip or sequence, the host sends *playmod_EnterScrub*, indicating that a series of positioning calls is about to come, followed by a series of *playmod_SetPos* commands. The player should position itself to the location described by *playmod_SetPos*, display a frame of video, and play a blip of audio. The host ends the sequence with *playmod_LeaveScrub*. When the user steps back and forth using the arrow keys, the host sends *playmod_EnterScrub*, *playmod_Step*, *playmod_LeaveScrub*.

When the user initiates playback, the host sends *playmod_Preroll* to tell the player to prepare for playback. The player should ready the output device, the audio hardware using the Playmod Audio Suite, etc. The host then sends *playmod_Play* to begin playback. If the host is playing the audio through an ASIO driver (recommended), the player should call the host to get the current audio position, to use as a clock for the video. If the player is driving the audio, it should call the host to get mixed-down audio buffers. The host repeatedly sends *playmod_GetPos* to retrieve the player's current position and state (playing or stopped). The host sends *playmod_PlayIdle* to give the player processor time to service the playback. The player must update the screen and/or the external device as quickly as possible, either rendering the frames itself, or calling the Playmod Render Suite to render the frames. If playback is stopped by the user, *playmod_Stop* is sent. Otherwise, the player should stop itself when it reaches the out point specified in *playmod_Play*, or in point if playing backwards, or loop back to the in/out point if looping was specified in *playmod_Play*.

If the user closes the project, the host sends *playmod_Close*. Release any hardware or memory.

playmod_Shutdown is sent when the host terminates.

Try the Sample Player Plug-in

A player requires a little more setup than most other plug-ins:
1) First build the player into the plug-ins folder
2) Build the exporter into the same folder
3) Players must have an Editing Mode XML to link it with an exporter to form an editing mode. Copy the `SDK Editing Mode.xml` file from the `Examples\Editing Modes\` folder in the SDK to the `Editing Modes` subfolder in the application installation folder.
4) Players also need Sequence Encoder Presets. Copy the folder in Examples\Projects\RTPlayback\SequencePreview\ to the folder described here.
5) Launch Premiere Pro, and in a new or existing project, create a New Sequence. In the New Sequence dialog, in the General tab, set the Editing Mode to PlayerSDK.

Real-time or Needs Rendering?

A player must analyze each segment of the cutlist, and tell the host whether or not the segment needs to be rendered to render files to achieve playback at the full frame rate. For example, the DV player used in the DV Editing Mode tells the host that a segment is real-time if it includes only one visible DV clip that matches the frame size and aspect ratio of the sequence. If more than one DV clip is used (such as a semi-transparent clip over another, or a Picture-In-Picture effect), or if any non-DV clip is used (such as a QuickTime clip, or a Premiere title), or if any effect is used, the DV player tells the host that the segment is not real-time. The host then marks the segment non-real-time using a red bar above the segment in the Sequence Panel.

All non-real-time segments must be rendered before a sequence can be Exported To Tape. Render files are created in the format specified by the Editing Mode, in a format supported by the exporter in the editing mode. Logically, the exporter should create render files in a format that the player can play at the full frame rate without dropping frames.

Which Pixel Formats to Use?

Since the player drives any rendering done during playback, the player must provide an ordered list of pixel formats as one of the render parameters. This list of pixel formats allows the renderer to decide how best to perform the rendering, based on the pieces of the segment, and the other parameters of the render request.

New in CS5, the player can use a combination of calls to get any format desired. It is never guaranteed that the built-in renderers will return a specific format, since it will use the shortest conversion path. However, `ScaleConvert()`, in the new Image Processing Suite, will convert any of the supported formats into any other.

So going back to the list of pixel formats, we recommend you request two:

1) Some 4:4:4:4 format – The built-in renderer will use this to choose a pixel format when it absolutely needs to make a conversion and the choice is arbitrary. It also uses this to decide how deep to do some processing. For example, if the player asks for `VUYA_4444_8u`, and the renderer needs to make a framerate adjustment, then it will do that work in a 8-bit Y'UV space. If the player asks for `VUYA_4444_32f`, then wherever possible, the renderer will process in 32-bit. In the built-in player, this is controlled in the Sequence Settings by the "Maximum Bit Depth" checkbox.

2. `PrPixelFormat_Any` - This tells the built-in renderers that any format is acceptable and that it should avoid unnecessary conversions. If the player receives back a format that it isn't expecting or can't handle, then it can use `ScaleConvert()` in the new Image Processing Suite to convert to the buffer into any format desired.

If a player wants a compressed or subsampled format after the render, it can use the Image Processing Suite to convert. If using asynchronous rendering, it can even do the conversion in a completion callback, so that it happens on multiple threads.

Why Can't I Always Get a Compressed Frame Back?

If the built-in renderer has to do any processing, then it will do so in a 4:4:4:4 format. As a general rule, in a render it will never post-encode to a compressed or subsampled format - once it gets to 4:4:4:4, it will stay there to preserve the quality.

So the renderer won't return a compressed frame if the source footage has a different compression or is uncompressed. If the source frame has the same compression, and the player still doesn't get

that format from the renderer, then it implies that the renderer is doing some kind of processing on the frames. There must be a mismatch in frame size, PAR, field type, frame rate, etc.

Segments

A cutlist is a list of tree structures of video clips and effects. Each tree structure represents a segment of the sequence. The following is an example segment:

```
Segment
     Compositor Node
          Clip Node
               Media Node (bottom clip)
          Clip Node
               Media Node (middle clip)
          Clip Node
               Media Node (top clip)
               Clip Operators (Blur, Color Corrector, Motion)
```

To get a good idea of the segment structure, try the SDK player on Windows, create a sequence using the SDK Editing Mode, and watch the text overlay in the Sequence Monitor as you perform edits.

High-Bit Color Depth

In the Sequence Settings, in the Video Previews area, the Maximum Bit Depth checkbox allows the user to toggle between 8-bit color rendering for speed, and the deepest supported color rendering for quality. This setting corresponds to the `useMaximumRenderPrecision` member of `pmPlaySettings`, sent with the *playmod_PushPlayerSettings* selector.

Multi-Camera Monitor

Players can not assume that the video display area matches the sequence dimensions. If the player is created for a Multi-Camera Monitor, it can be larger than the sequence dimensions. In the Multi-Camera Monitor, effect parameters can change during playback, so the rendering of the effects should occur at the last possible moment. The player will be sent the new playmod_EnableDynamicPlayback to tell it to enter/exit dynamic playback mode.

Real-Time Titling and Stills

For real-time titles and still images, during *playmod_VideoSequenceHasChanged*, use the Clip Render Suite to render the title if the clip is a still, save the rendered image, and display it at the appropriate time during playback.

What About Audio?

All audio rendering is done internally by the host. The player controls audio playback using the Playmod Audio Suite. Segments do not contain any information on the audio in the sequence.

Entry Point

```
int xPlayEntry (
     int             selector,
     pmStdParms*     stdParms,
     void*           param1,
     void*           param2)
```

selector is the action the host wants the player to perform. `stdParms` provides callbacks to obtain additional information from the host or to have the host perform tasks. Parameters 1 and 2 vary with the selector; they may contain a specific value or a pointer to a structure. Return `playmod_ErrNone` if successful, an appropriate return code, or `playmod_Unsupported` if an unrecognized selector is sent. There are other specific responses to some selectors that aren't errors.

PrPlayID

Many selectors pass a `PrPlayID` as param1. The `PrPlayID` is a pointer to a private structure defined, created, and managed solely by an instance of the player, and is passed to most selectors for the player to use as storage. It should be created during *playmod_NewList*, and disposed of during *playmod_Close*. The player is responsible for all memory management in this structure.

Standard Parameters

A pointer to this structure is sent from the host to the player with every selector.

```
typedef struct {
     int                 pmInterfaceVer;
     pmCallbackFuncs     *funcs;
     piSuitesPtr         piSuites;
} pmStdParms;
```

Member	Description

`pmInterfaceVer`	Player API version Premiere Pro CS5 - `PLAYMOD_VERSION_110` Premiere Pro CS4 - `PLAYMOD_VERSION_100` Premiere Pro CS3 - `PLAYMOD_VERSION_90`
`funcs`	Pointers to callbacks for players.
`piSuites`	Pointer to universal callback suites.
`playmodPrefs`	Pointer to private player preferences data (saved in the project file). Data is returned to the plug-in with every subsequent call. See *playmod_GetFilePrefs*.

Player-Specific Callbacks

```
typedef struct {
     ClassDataFuncsPtr        classFuncs;
     FileFuncsPtr             fileFuncs;
     VideoFuncsPtr            videoFuncs;
} pmCallbackFuncs;
```

Callbacks	Description
`classFuncs`	See ClassData functions
`fileFuncs`	File Callbacks.
`videoFuncs`	Video Callbacks.

File Callbacks

Only `getPixelAspectRatio` is not deprecated.

```
typedef struct {
     pmOpenFileFunc                openFile;
     pmReleaseFileFunc             releaseFile;
     pmSetDebugParameterFunc       setDebugParameter;
     pmGetPixelAspectRatioFunc     getPixelAspectRatio;
} FileFuncs, *FileFuncsPtr;
```

Function	Description
`openFile, releaseFile, setDebugParameter`	Deprecated
`getPixelAspectRatio`	Used to get the pixel aspect ratio of a file. No longer needed in CS3 and later. Use the Video Segment Suite instead.

getPixelAspectRatio

Used to get the pixel aspect ratio of a file.

```
void getPixelAspectRatioFunc(
     PrPlayID       playID,
     csSDK_uint32   *num,
     csSDK_uint32   *den);
```

Parameter	Description
playID	The ID of the player instance. Provided by the host.
num	The pixel aspect ratio, represented by a rational value, with a numerator and a denominator.
den	

Video Callbacks

Many of these callbacks are obsolete, or superceded by the Playmod Render Suite.

```
typedef struct {
     pmShowFileFrameFunc            showFileFrame;
     pmShowFileFrameProxyFunc       showFileFrameProxy;
     pmShowNeedsRenderXFunc         showNeedsRenderX;
     pmShowFileFrameOffscreenFunc   showFileFrameOffscreen;
     pmGetCurrentTime               getCurrentTime;
     pmFrameDropped                 frameDropped;
     pmShowFileFrameWithSafeAreasFunc showFileFrameWithSafeAreas;
     pmShowFileFrameRenderSettings  showFileFrameRenderSettings;
} VideoFuncs, *VideoFuncsPtr;
```

Function	Description
showFileFrame	Windows only. Tells the host to render and display a frame of video in the Monitor.
showFileFrameProxy, showNeedsRenderX, showFileFrameOffscreen	Obsolete. Calling these functions has no effect.
getCurrentTime	Get the current time during playback, as determined by the audio clock.
frameDropped	Reports dropped frames during Export To Tape.
showFileFrameWithSafeAreas	Tells the host to render and display a frame of video in the Monitor, with safe areas overlaid.

showFileFrameRenderSettings	New in Premiere Pro 2.0. Sets additional parameters for showFileFrame.

showFileFrame

Tells the host to render and display a frame of video in the Monitor. If you're playing a cutlist and encounter frames you cannot play, call this function to ask the host to display the frame. If frames must be sent to external hardware, or more control is needed over render parameters, use RenderVideoFrame in the Playmod Render Suite.

```
void showFileFrameFunc(
     PrPlayID       playID,
     csSDK_int32    frametime,
     pmDisplayPos   *disp,
     prFloatRect    *view);
```

Parameter	Description
playID	The ID of the player instance. Provided by the host.
frametime	Specifies the frame to render and display.
disp	Specifies where in the window the video should be displayed. This is passed to the player during *playmod_SetDisp*.
view	Specifies the area of the video that should be displayed. This is passed to the player during playmod_SetView.

getCurrentTime

Get the current time during playback, as determined by the audio clock.

```
int getCurrentTime(
     PrPlayID       playID,
     double         *inCurrentTime);
```

Parameter	Description
playID	The ID of the player instance. Provided by the host.
currentTime	Returned by the callback. The value is in milliseconds.

frameDropped

Reports dropped frames during Export To Tape, when using the standard Export To Tape dialog. The host keeps track of the total number of dropped frames reported with this callback, displays

the number to the user, and aborts when the user-defined limit is reached.

```
void frameDropped(
     PrPlayID       playID,
     csSDK_int32    inNumberFramesDropped);
```

Parameter	Description
playID	The ID of the player instance. Provided by the host.
inNumberFramesDropped	Specifies the number of frames dropped.

showFileFrameWithSafeAreas

Tells the host to render and display a frame of video in the Monitor, with safe areas overlaid. If you're playing a cutlist and encounter frames you cannot play, and safe areas are on, call this function to ask the host to display the frame. If frames must be sent to external hardware, or more control is needed over render parameters, use RenderVideoFrame in the Playmod Render Suite.

```
void showFileFrameFuncWithSafeAreas(
     PrPlayID                playID,
     csSDK_int32             frametime,
     pmDisplayPos            *disp,
     prFloatRect             *view,
     pmAdornSafeAreasParams  *adornSafeAreaParams);
```

Parameter	Description
playID	The ID of the player instance. Provided by the host.
frametime	Specifies the frame to render and display.
disp	Specifies where in the window the video should be displayed. This is passed to the player during *playmod_SetDisp*.
view	Specifies the area of the video that should be displayed. This is passed to the player during *playmod_SetView*.
adornSafeAreaParams	Specifies the details of the safe area that should be displayed. This is passed to the player during *playmod_AdornSafeAreas*

showFileFrameRenderSettings

New in Premiere Pro 2.0. Sets additional parameters for `showFileFrame`.

```
void showFileFrameRenderSettings(
     PrPlayID                playID,
     const int               inWidth,
```

```
const int                   inHeight,
const PrRenderQuality       inRenderQuality,
const prBool                inRenderFields);
```

Parameter	Description
playID	The ID of the player instance. Provided by the host.
inWidth	Sets the dimensions to be used by showFileFrame.
inHeight	
inRenderQuality	Sets the render quality.
inRenderFields	Sets the field rendering setting.

Selector Table

This table summarizes the various selector commands a player can receive.

Selector Messaging Selectors	param1	param2
playmod_Startup	pmStartupRec*	unused
playmod_Shutdown	unused	unused
playmod_GetIndFormat	pmModuleInfoRec*	(int) index
playmod_GetInfo	PrPlayID	pmPlayInfoRec*
playmod_GetFilePrefs	PrPlayID	pmGetFilePrefsRec*
playmod_SetFilePrefs	PrPlayID	void *playmodPrefs
playmod_PushPlayerSettings	PrPlayID	pmPlayerSettings*
playmod_Open	No longer sent in CS3 and later	
playmod_Close	PrPlayID	unused
playmod_Activate	PrPlayID	pmActivateRec*
playmod_Update	PrPlayID	unused
playmod_UpdateMarkers	unused	unused
playmod_SetDisp	PrPlayID	pmDisplayPos*
playmod_SetView	PrPlayID	prFloatRect*
playmod_SetDisplayMode	PrPlayID	pmDisplayMode
playmod_SetVideoDisplayType	PrPlayID	PrVideoDisplay-Parameters*
playmod_SetDisplayStateProperties	PrPlayID	pmDisplayStateProper-ties*
playmod_SetQuality	PrPlayID	PrPlaybackQuality
playmod_SetUseFractionalResolution	long	unused

playmod_SetFractionalResolution	long	unused
playmod_AdornSafeAreas	PrPlayID	pmAdornSafeAreas-Params
playmod_ProjectSettingsChanged	PrPlayID	unused
playmod_DisplayMoving	PrPlayID	prRect*
playmod_DisplayChanged	PrPlayID	unused
playmod_GetAudioInfo	pmAudioInfo *	unused
playmod_GetAudioChannelInfo	pmAudioChan-nelInfo *	unused
playmod_EnableDynamicPlayback	prBool	unused
Playback Selectors		
playmod_GetPos	PrPlayID	pmGetPosRec*
playmod_Preroll	PrPlayID	pmPlayParms*
playmod_Play	PrPlayID	pmPlayParms*
playmod_PlayIdle	PrPlayID	unused
playmod_SetPlaybackSpeed	PrPlayID	float *inoutPlaybackSpeed
playmod_Stop	PrPlayID	unused
playmod_EnterScrub	PrPlayID	prBool
playmod_SetPos	PrPlayID	PrTime *
playmod_Step	PrPlayID	pmStepRec*
playmod_LeaveScrub	PrPlayID	unused
playmod_PutTemporaryTimeline	PrPlayID	pmPutTemporaryTimeli-neRec*
playmod_PutFrameRequest	PrPlayID	pmPutFrameRequestRec*
playmod_PutFrame	PrPlayID	pmPutFrameRec*
playmod_AllowSetPositionDuringPlayback	Not used in Premiere Pro or Encore. Only used in Soundbooth for the built-in player.	
Video Segment Selectors		
playmod_NewList	PrPlayID	pmNewListParms*
playmod_VideoSequenceHasChanged	PrPlayID	unused
playmod_GetRTStatusForTime	PrPlayID	prtPlayableRangeRec*

Selector Descriptions

This section provides a brief overview of each selector and highlights implementation issues.

playmod_Startup

```
param1 - pmStartupRec *
param2 - unused
```

Sent when the host launches. Determine if your plug-in has the necessary software and/or drivers to play the files it supports. Provide the plug-in specified player ID in `pmStartupRec`. For the player to be used in an editing mode, this ID must match the `<EditingMode.Player>` element in an editing mode XML file. If the call returns any error, the host will not call this player again. Heartless, we know.

playmod_Shutdown

```
param1 - unused
param2 - unused
```

Sent when the host terminates. Release any hardware drivers, and free all memory. Only `stdParms` is sent with this selector.

playmod_GetIndFormat

```
param1 - pmModuleInfoRec *
param2 - (int) index
```

Populate `pmModuleInfoRec`, describing which filetypes and subtypes the module supports. You'll continue to get this selector so you can iterate through all the filetypes you support, until you return `playmod_BadFormatIndex`. See Additional Details for more information.

playmod_GetInfo

```
param1 - PrPlayID
param2 - pmPlayInfoRec *
```

Return information about the file in `PrPlayID` in the `pmPlayInfoRec`.

playmod_GetFilePrefs

```
param1 - PrPlayID
param2 - pmGetFilePrefsRec *
```

Sent whenever the user selects Playback Settings in the Sequence > Sequence Settings > General dialog. Display a settings dialog, populated either with defaults or using data from the `pmGetFilePrefsRec`. If the changes made should not force a rebuild of the cutlist, return `playmod_ErrNone`, otherwise return `playmod_RebuildCutlist`. Return `playmod_BroadcastPrefs` if prefs have changed so that the host can call all `playmod_SetFilePrefs` on all open players with the updated prefs.

playmod_SetFilePrefs

```
param1 - PrPlayID
param2 - void *playmodPrefs
```

Notification to a player that the Playback Settings in the Sequence > Sequence Settings > General dialog have changed and should be updated accordingly.

playmod_PushPlayerSettings

```
param1 - PrPlayID
param2 - pmPlayerSettings *
```

New in Premiere Pro 2.0. Notification to a player that the general playback settings have changed and should be updated accordingly.

playmod_Close

```
param1 - PrPlayID
param2 - unused
```

Release any handles or open files, dispose any memory still allocated.

playmod_Activate

```
param1 - PrPlayID
param2 - pmActivateRec
```

Sets the activation state. At any time there can be more than one player instance, but only one is active. Activation means the plug-in may be playing soon. Player instances still receive *playmod_Update* and *playmod_SetPos* to redraw windows while deactivated, but no messages to begin playback will be sent to it until the player instance is activated.

On deactivate, release hardware resources, sound channels, memory so other modules can play. On activate, start all driver/play hardware, do not update current frame until *playmod_Update* message is sent.

playmod_Update

```
param1 - PrPlayID
param2 - unused
```
Redraw the current frame (the playback area was obscured by another window and has now been uncovered).

playmod_UpdateMarkers

```
param1 - unused
param2 - unused
```

The timeline markers have changed; use `stdParms->piSuites->timelineFuncs` to retrieve new marker data. Markers are read-only, and cannot be changed by plug-ins.

playmod_SetDisp

```
param1 - PrPlayID
param2 - pmDisplayPos *
```

Describes the playback position on the screen. Save this information for use during *playmod_Play*.

playmod_SetView

```
param1 - PrPlayID
param2 - prFloatRect *
```

Describes the viewable area of the video in the monitor, in normalized coordinates. For example, if the entire video is viewable, the top and left will be zero, and the bottom and right will be one. If the monitor is zoomed into the video, only a part of the video will be visible. Save this information for use during playback.

playmod_SetDisplayMode

```
param1 - PrPlayID
param2 - pmDisplayMode
```

Obsolete. Replaced by *playmod_SetVideoDisplayType* in Premiere Pro 1.5. Sets the current display mode of the player. Display modes include:
`pmPlayerDisplay_Off`,
`pmPlayerDisplay_Composite` - The standard composite video mode,
`pmPlayerDisplay_Alpha` - Alpha mode,
`pmPlayerDisplay_Scope` - Scopes display mode,
`pmPlayerDisplay_DirectManipulation` - Direct manipulation mode.
`pmPlayerDisplay_AudioOnly` - Audio-only clip in the Source Monitor.

playmod_SetVideoDisplayType

```
param1 - PrPlayID
param2 - PrVideoDisplayParameters *
```

Sets the current display parameters of the player. This tells the player whether it should be displaying the composite video, just playing audio, or in one of the scopes modes. If in one of the scopes modes, the scopes settings are also provided.

A player isn't required to draw scopes. To draw the scopes to an external output, then yes, it must handle that drawing itself. Otherwise, it can choose to just support `kPrVideoDisplayType_Composite`, `kPrVideoDisplayType_Off`, and `kPrVideoDisplayType_AudioOnly`. Premiere Pro will handle drawing the rest of the types to the screen if the player returns `playmod_Unsupported` for any other display type.

When playing an audio-only clip in the Source Monitor, turn off video display, and don't round the position returned in *playmod_GetPos* to video frame boundaries. This allows for smoother current time indicator movement, and prevents the CTI from jumping back to the previous video frame boundary when playback begins.

playmod_SetDisplayStateProperties

```
param1 - PrPlayID
param2 - pmDisplayStateProperties*
```

New in CS5, and supercedes *playmod_SetQuality*, *playmod_SetUseFractionalResolution*, and *playmod_SetFractionalResolution*. Premiere Pro passes display settings to the player, as set by the user in the Monitor panel settings.

playmod_SetQuality

```
param1 - PrPlayID
param2 - PrPlaybackQuality
```

Superceded by *playmod_SetDisplayStateProperties* in CS5. Set/change playback quality. This quality is set by the user in the Monitor settings. It is not changed during playback. Each player instance can have its own state.

playmod_SetUseFractionalResolution

```
param1 - PrPlayID
param2 - long
```

Superceded by *playmod_SetDisplayStateProperties* in CS5. New in Premiere Pro 4.1. Turn on or off fractional resolution. param2 contains a boolean, so if it is non-zero, the user has turned on fractional resolution mode. This quality is set by the user in the Monitor settings. It is not changed during playback. Each player instance can have its own state.

playmod_SetFractionalResolution

```
param1 - PrPlayID
param2 - long
```

Superceded by *playmod_SetDisplayStateProperties* in CS5. New in Premiere Pro 4.1. Set the current fractional resolution mode. param2 contains the enumerated value. This quality is set by the user in the Monitor settings. It is not changed during playback. Each player instance can have its own state.

playmod_AdornSafeAreas

```
param1 - PrPlayID
param2 - pmAdornSafeAreasParams
```

Provides attributes for safe title and safe action overlays

playmod_ProjectSettingsChanged

```
param1 - PrPlayID
param2 - unused
```

At least some of the project settings have changed. Use our timeline and utility funcs to determine whether or not you need to rebuild the cutlist or create new preview files.

playmod_DisplayMoving

```
param1 - PrPlayID
param2 - prRect *
```

Notifies the player that a window is moving. Assuming the window is not changing size the same time, the rect passed in is likely to stay the same. *playmod_SetDisp* is never called when the window is moving and hence this new selector is used for that. Most old players don't really need to know when the window is moving since Windows blits the images around.

playmod_DisplayChanged

```
param1 - PrPlayID
param2 - unused
```

Notifies a player that the desktop display has changed, likely do to a color depth or resolution change. It's useful when players need to know this in order to optimizing their code that displays their frames on the desktop.

playmod_GetAudioInfo

```
param1 - pmAudioInfo *
param2 - unused
```

New in Premiere Pro 2.0. Gets information about the audio capabilities of the player. This selector should be implemented if the player uses plug-in based audio, calling InitPluginAudio in version 2 or higher of the Playmod Audio Suite. This provides the host with information for the Preferences > Audio Output Mapping dialog, where the user can assign channel mappings for audio output.

playmod_GetAudioChannelInfo

```
param1 - pmAudioChannelInfo *
param2 - unused
```

New in Premiere Pro 2.0. Gets information about a specific audio channel of the player. This selector should be implemented if the player uses plug-in based audio, calling InitPluginAudio in version 2 or higher of the Playmod Audio Suite. This provides the host with information for the Preferences > Audio Output Mapping dialog, where the user can assign channel mappings for audio output.

playmod_EnableDynamicPlayback

```
param1 - prBool
param2 - unused
```

New in Premiere Pro 2.0. Gives a hint to help playback in the Multicam Monitor. If this is enabled, a player should expect that effect parameters can change during playback, so the rendering of the effects should occur at the last possible moment.

playmod_GetPos

```
param1 - PrPlayID
param2 - pmGetPosRec *
```

Return the current position and mode to the host. Fill in the position and current mode (playmode_Playing, or playmode_Stopped) in `pmGetPosRec`.

playmod_Preroll

```
param1 - PrPlayID
param2 - pmPlayParms
```

Prepare a player to playback. Sent immediately before *playmod_Play*. The player should ready the output device, the audio hardware using the Playmod Audio Suite, etc. The `pmPlayParams` describe where to play and stop.

playmod_Play

```
param1 - PrPlayID
param2 - pmPlayParms *
```

Begin playback. The members of the `pmPlayParms` structure describe the position to start at, in/out point, loop flag, and speed.

playmod_PlayIdle

```
param1 - PrPlayID
param2 - unused
```

Sent repeatedly during play to give the playback module time to service the playback. The player should stop itself when it reaches the out point specified in *playmod_Play*, or in point if playing backwards, or loop back to the in/out point if looping was specified in *playmod_Play*.

playmod_SetPlaybackSpeed

```
param1 - PrPlayID
param2 - float *inoutPlaybackSpeed
```

A request that the player change to the requested playback speed. The speed value is a float, with 1.0 meaning forward play at normal speed and (-1.0) meaning backward play at normal speed. Zero is an illegal speed an will never be passed down. However, values very close to zero can be passed down. The player can refuse to play at a requested speed, in which case the player should change the float value to the speed it is going to play at and return the new *playmod_UnsupportedPlaybackSpeed* error code.

playmod_Stop

```
param1 - PrPlayID
param2 - unused
```

Stop playback. This selector is only sent when playback is stopped by the user. Otherwise, the player should stop itself when it reaches the output specified in *playmod_Play*, or in point if playing backwards, or loop back to the in/out point if looping was specified in *playmod_Play*.

playmod_EnterScrub

```
param1 - PrPlayID
param2 - prBool
```

The user has started scrubbing, and a series of *playmod_SetPos* calls should follow. If the second parameter is false, the playmod should initialize the audio system and keep it initialized until *playmod_LeaveScrub* is called. On *playmod_SetPos* calls, it should move to the new position and play a blip of audio. If the second parameter is true, then any audio from the host will be silent and there is no need to initialize the audio system.

playmod_SetPos

```
param1 - PrPlayID
param2 - PrTime *
```

Sent when the user scrubs through a clip or sequence. Jump to the position specified at `PrTime`, and display the frame at the new position. If player is scrubbing, play an audio blip using the Playmod Audio Suite. Can be sent when player is not in scrub mode, for example, when scrubbing a timecode hottext.

playmod_Step

```
param1 - PrPlayID
param2 - pmStepRec *
```

Move the current position by the requested time, and display the frame at the new position. If player is scrubbing, play an audio blip using the Playmod Audio Suite.

playmod_LeaveScrub

```
param1 - PrPlayID
param2 - unused
```

The user has finished scrubbing. If the audio system is active, the player should uninitialize it.

playmod_PutTemporaryTimeline

param1 - `PrPlayID`
param2 - `pmPutTemporaryTimelineRec*`

New in CS5. Optional. A player should support this selector it wants to render a frame of a temporary timeline and display it on the Monitor panel and any video output destinations. This is a replacement to the old *playmod_PutSegment* functionality. If the player wants to display the frame, but doesn't need to handle the rendering itself, it should support *playmod_PutFrame* instead. This selector is called only if the player set `PMCapCanPutTemporaryTimeline` in `pmModuleInfoRec.capabilityFlags`.

playmod_PutFrameRequest

```
param1 - PrPlayID
param2 - pmPutFrameRequestRec *
```

Called only after `playmod_RenderAndPutFrame` was returned from *playmod_PutSegment*, when a segment is available and can be rendered at the preferred size of the player. Sent to query for the frame size and pixel format the player would like to receive in a subsequent call to *playmod_PutFrame*. *playmod_PutFrame* is not always preceeded by *playmod_PutFrameRequest*.

If `playmod_ErrNone` is not returned, *playmod_PutFrame* will not be sent, and the host will update only the Monitor. Returning an error code does not prevent the player from receiving subsequent calls of *playmod_PutFrameRequest*.

playmod_PutFrame

```
param1 - PrPlayID
param2 - pmPutFrameRec *
```

Passes the host-rendered frame to the player for display on the external device and/or the screen. The player should display the frame on the external device and draw the frame on the screen. This is used for updating the video display when performing and edit, when frames are output from the titler, during direct manipulation (Motion, Transform, and other effects), and other scenarios where only a rendered frame is available. If the player wishes to perform its own rendering for this frame, it should implement *playmod_PutTemporaryTimeline* instead.

This selector is called only if the `PMCapPutFrame` flag is set for the player. If *playmod_PutFrameRequest* is sent first, the size of the frame that is specified by the player will be the size sent to *playmod_PutFrame*. If *playmod_PutFrameRequest* is not sent, any size image can be sent to *playmod_PutFrame*.

Prior to CS5, the player had to return `playmod_OverlayDone` if the it had displayed the frame on the screen so that the host will not draw over the results. On Windows only, the player could return `playmod_ErrNone` if it had displayed the frame on the external device only, and the host would draw the frame on the screen.

playmod_NewList

```
param1 - PrPlayID *
param2 - pmNewListParms *
```

Initialize a new player instance, with a unique cutlist.

There can be multiple instances of the same sequence. For example, when the user opens a sequence, one instance is created. When the titler is first opened, another instance will be created to provide the background video. If a transition is selected in the sequence, and the Effect Controls Panel is open, another instance will be created to provide the transition preview.

Private data stored in `PrPlayID` will be returned to the plug-in with all subsequent selectors. `pmNewListParms` describes the lists' timebase and screen position.

playmod_VideoSequenceHasChanged

```
param1 - PrPlayID
param2 - unused
```

A video sequence has been updated. An edit was made to the timeline that affects the output. If the player takes over rendering in at least certain cases, the player should use the Video Segment Suite to parse the timeline, and reevaluate which areas of the timeline are real-time, and which areas are not. The real-time status should be returned to the host later during *playmod_GetRTStatusForTime*.

For RT titling and still images, use the Clip Render Suite to render the title if the clip is a still, save the rendered image, and play it back at the appropriate time during playback.

playmod_GetRTStatusForTime

```
param1 - PrPlayID
param2 - prtPlayableRangeRec
```

Give the host the status of the current cutlist. If a specific segment has previously been added to the cutlist and was acknowledged in its entirety (can be played intact and in Real-time), it is said to be a "Real-time" segment. A segment that plays at a lower frame rate during playback, or is "scaled down" to remove elements that are not supported by the player is not Real-time".

A non-RT segment is marked with a red line on the timeline.

Return Codes

Return Code	Reason
playmod_ErrNone	Operation has completed without error.
playmod_ErrBadFile	File is corrupt or unreadable.
playmod_ErrDriver	A driver error is detected, perhaps because of a hardware failure.
playmod_ErrNotPreferred	The file subtype is incorrect for this module.
playmod_BadFormatIndex	The format index is invalid (for *playmod_GetIndFormat*).
playmod_DeclinePlay	Return this if your player declines to play this clip, the host will attempt to find another player which will.
playmod_ListWrongType	Player can't play this clip, it is the wrong type. This asks the host to render it.
playmod_ListBadSpeed	Module can't play back this file at the requested speed.

playmod_CantAddSegment	The cutlist can't add a segment.
playmod_Unsupported	Always return this if you're sent a selector that you don't support.
playmod_AudioOverload	Unused.
playmod_OutOfRange	Unused.
playmod_CannotRender	The plug-in cannot render the frame in real-time.
playmod_RebuildCutlist	Return value used in *playmod_GetFilePrefs* to force the cutlist to be rebuilt.
playmod_CannotShiftLayer	Unused.
playmod_OverlayDone	No longer supported in CS5. Only supported on Windows in CS3 and CS4. Return value used in *playmod_PutFrame* to tell the host that the player has successfully updated the Monitor, so that the host will not draw over the results.
playmod_UnsupportedPlaybackSpeed	Return from *playmod_SetPlaybackSpeed* if the requested speed is not supported.
playmod_BroadcastPrefs	Return from *playmod_GetFilePrefs* to have the host call *playmod_SetFilePrefs* on all open players to update them with using the returned prefs.
playmod_CannotRecord	Return from *playmod_Play* if audio voiceover recording is requested and the player cannot support it. The audio mixer will then put up a descriptive error message.
playmod_RenderAndPutFrame	Unused
pmIsCacheable	Return from *playmod_Startup* if the player is cacheable, or playmod_ErrNone if not cacheable.

Structures

Structure	Sent with selector
Messaging Structures	
pmActivateRec	*playmod_Activate*
pmAdornSafeAreasParams	*playmod_AdornSafeAreas*
pmAudioChannelInfo	*playmod_GetAudioChannelInfo*
pmAudioInfo	*playmod_GetAudioInfo*
pmDisplayPos	*playmod_NewList* (member of pmNewListParms), *playmod_SetDisp*. Also used in showFileFrame and showFileFrameWithSafeAreas callbacks.
pmDisplayStateProperties	*playmod_SetDisplayStateProperties*

pmFileSpec	No longer used in CS4
pmGetFilePrefsRec	*playmod_GetFilePrefs*
pmModuleInfoRec	*playmod_GetIndFormat*
pmPlayerSettings	*playmod_PushPlayerSettings*
pmPlayInfoRec	*playmod_GetInfo*
pmPlayOpenParms	No longer used in CS4
PrPlayID	Almost all selectors
pmPlayTimebase	*playmod_NewList* (member of pmNewListParms)
pmPutFrameRec	*playmod_PutFrame*
pmPutFrameRequestRec	*playmod_PutFrameRequest*
pmPutTemporaryTimeli-neRec	*playmod_PutTemporaryTimeline*
pmStartupRec	*playmod_Startup*
pmViewRec	*playmod_SetView*
PrVideoDisplayParameters	*playmod_SetVideoDisplayType*
Playback Structures	
pmGetPosRec	*playmod_GetPos*
pmPlayParms	*playmod_Preroll* and *playmod_Play*
pmStepRec	*playmod_Step*
Cutlist Structures	
pmNewListParms	*playmod_NewList*
prtPlayableRangeRec	*playmod_GetRTStatusForTime*

Structure Descriptions

pmActivateRec

Selector: *playmod_Activate*

Provides application activation information.

```
typedef struct {
     prBool              activate;
     PrActivationEvent   activationEvent;
     PrFourCC            pluginClassID;
     PrFourCC            pluginFileType;
} pmActivateRec;
```

activate	If true, and the player is not active, activate it. If false, and the player is not inactive, deactivate it.
activationEvent	The type of event causing the activation/deactivation: PrActivationEvent_Unspecified, PrActivationEvent_RecorderActivated, PrActivationEvent_PlayerActivated, or PrActivationEvent_ApplicationLostFocus
pluginClassID	
pluginFileType	

pmAdornSafeAreasParams

Selector: *playmod_AdornSafeAreas*

Attributes for safe title and safe action overlays.

```
typedef struct {
     prBool          enableSafeTitle;
     float           safeTitlePercentWidth;
     float           safeTitlePercentHeight;
     prBool          enableSafeAction;
     float           safeActionPercentWidth;
     float           safeActionPercentHeight;
} pmAdornSafeAreasParams;
```

enableSafeTitle	If true, safe title should be overlaid.
safeTitlePercentWidth	How far (percent-wise) to inset the overlaid rectangle displaying safe title (i.e. for a 720x480 display, "0.1" would mean to inset the rectangle by 72x48 pixels)
safeTitlePercentHeight	
enableSafeAction	If true, safe action should be overlaid.
safeActionPercentWidth	How far (percent-wise) to inset the overlaid rectangle displaying safe action.
safeActionPercentHeight	

pmAudioChannelInfo

Selector: *playmod_GetAudioChannelInfo*

Describes a specific audio channel of the player.

```
typedef struct {
     PrMemoryPtr     inPrefs;
```

```
     unsigned int   inPrefsSize;
     unsigned int   inChannelIndex;
     int            inIsInput;
     prUTF16Char    outChannelName[256];
} pmAudioChannelInfo;
```

inPrefs	A pointer to the player preference data.
inPrefsSize	Size of preference data.
inChannelIndex	The index of the channel for which to return the info.
inIsInput	If set to true, then the request is for an input channel. Otherwise, the request is for an output channel.
outChannelName	Set this to the Unicode name for the channel.

pmAudioInfo

Selector: *playmod_GetAudioInfo*

Describes the audio capabilities of the player.

```
typedef struct {
     PrMemoryPtr    inPrefs;
     unsigned int   inPrefsSize;
     unsigned int   outNumInputChannels;
     unsigned int   outNumOutputChannels;
     int            outWillUsePluginAudio;
     prUTF16Char    outAudioDisplayName[256];
} pmAudioInfo;
```

inPrefs	A pointer to the player preference data.
inPrefsSize	Size of preference data.
outNumInputChannels	Currently unused.
outNumOutputChannels	The number of output channels for the player.
outWillUsePluginAudio	Set to true if the plug-in will call `InitPluginAudio`. If false, the call will fail if the player calls `InitPluginAudio`.
outAudioDisplayName	Set this to the Unicode display name of the current audio hardware.

pmDisplayPos

Selector: *playmod_NewList* (member of `pmNewListParms`) and *playmod_SetDisp*. Also used in `showFileFrame` and `showFileFrameWithSafeAreas` callbacks.

Describes the playback position on the screen where the movie should be displayed.

```
typedef struct {
     prWnd     wind;
     int       originTop;
     int       originLeft;
     int       originWidth;
     int       originHeight;
} pmDisplayPos;
```

wind	Display window.
originTop	Offset in pixels from top of display window.
originLeft	Offset in pixels from left of display window.
dispWidth	Width of display area in pixels.
dispHeight	Height of display area in pixels.

pmDisplayStateProperties

Selector: *playmod_SetDisplayStateProperties*

Describes the current video display quality and characteristics.

```
typedef struct
{
     pmPlayMode          inPlayMode;
     PrRenderQuality     inRenderQuality;
     PrRenderQuality     inDeinterlaceQuality;
     pmFieldDisplay      inFieldDisplay;
     csSDK_int32         inDownsampleFactor;
} pmDisplayStateProperties;
```

inPlayMode	Specifies the mode for to the rest of the settings apply. Either playmode_Stopped, playmode_Playing, or playmode_Scrubbing.
inRenderQuality	Render quality flag.
inDeinterlaceQuality	Render quality flag, applied to the deinterlacing quality.
inFieldDisplay	Either pmFieldDisplay_ShowFirstField, pmFieldDisplay_ShowSecondField, or pmFieldDisplay_ShowBothFields.

inDownsampleFactor	Factor at which to downsample the video. For example, with a factor of 2, the video can be rendered at half the sequence width and height for better performance.

pmGetFilePrefsRec

Selector: *playmod_GetFilePrefs*

```
typedef struct {
     PrFourCC       filetype;
     PrFourCC       subtype;
     PrFourCC       classID;
     PrMemoryPtr    *playmodPrefs;
     prBool         projectOpen;
} pmGetFilePrefsRec;
```

filetype	FourCC of filetype the preferences are requested for.
subtype	FourCC of subtype the preferences are requested for. Usually the codec identifier. It can also be 0.
classID	FourCC of class identifier. This is used to differentiate between compile modules that support the same filetype and to cross reference between different plug-in types (i.e. players and recorders).
playmodPrefs	A pointer to a structure, private to the player. If NULL, preferences data has not been stored and the player should allocate a block of memory using the Memory Functions. Otherwise, preferences have been already been stored and can be reused.
projectOpen	If set to true, this selector is being sent while a project is open. If false, there is currently no project open, for example, if the selector is being sent from the New Project dialog.

pmModuleInfoRec

Selector: *playmod_GetIndFormat*

Describes the capabilities of the player.

```
typedef struct {
     PrFourCC       filetype;
     PrFourCC       subtype;
     PrFourCC       classID;
     csSDK_int32    playflags;
     int            hasSetup;
```

```
    int             capabilityFlags;
    int             requestedAPIVersion;
    int             reserved[32];
} pmModuleInfoRec;
```

`filetype`	FourCC of filetype supported.
`subtype`	FourCC of subtype supported. Usually the codec identifier. It can also be 0.
`classID`	FourCC of class identifier. This is used to differentiate between compile modules that support the same filetype and to cross reference between different plug-in types (i.e. players and recorders).
`playflags`	Must be `pmFlag_canPlayLists`.
`hasSetup`	If non-zero, the host enables the Playback Settings button in Sequence > Sequence Settings > General. The player will receive *playmod_GetFilePrefs* when the user pushes this button, and you can display your setup dialog.

capabilityFlags	PMCapPutFrame Advertises the player can output frames to an external device using *playmod_PutFrame*. PMWillReportDroppedFrames Player will report dropped frames during Export To Tape. PMCapCanLoopPlayback Player can loop playback. PMCapCanShuttlePlayback Player can playback speeds between -4x and 4x. PMCapCanZoom Player supports different zoom levels in the Monitor. PMCapCanSafeMargin Player can overlay title and action-safe guides. PMCapCanExport Player supports Export To Tape. A device controller must also be active to establish a connection to the device. Otherwise, the menu item will be inactive. PMCapPutSegment Unused in CS4 and later. PMCapSingleClipPlayerWantsRTSegments Unused in CS3 and later. PMCapCanDoFractionalResolution New in CS4. Turn on fractional resolution capabilities for all editing modes supported by the player. PMCapCanPutTemporaryTimeline New in CS5. Advertises support for *playmod_PutTemporaryTimeline*. PMCapSupportsDisplayStateProperties New in CS5. Advertises support for *playmod_SetDisplayStateProperties*.
requestedAPIVersion	Unused.
reserved[32]	Do not use.

pmPlayerSettings

Selector: *playmod_PushPlayerSettings*

Notification to a player that the general playback settings have changed and should be updated accordingly.

```
typedef struct {
     int            useMaximumRenderPrecision;
     int            mSuppressTransmit;
} pmPlayerSettings;
```

useMaximumRenderPrecision	If set to true, the player should render at the highest bit depth possible.
mSuppressTransmit	If set to true, the player should only display the video on the desktop, not to any external device.

pmPlayInfoRec

Selector: *playmod_GetInfo*

Describes the clip or sequence to be played.

```
typedef struct {
     int            width;
     int            height;
     prBool         hasVideo;
     prBool         hasAudio;
     csSDK_int32    prefPreviewWidth;
     csSDK_int32    prefPreviewHeight;
     csSDK_uint32   pixelAspectNum;
     csSDK_uint32   pixelAspectDen;
} pmPlayInfoRec;
```

width	The host provides the dimensions of the clip or sequence
height	
hasVideo	True if clip has video. Always true for sequence players.
hasAudio	True if clip has audio. Always true for sequence players.
prefPreviewWidth	The host provides the dimensions of the preferred screen display. For now, this is always the same as width and height above.
prefPreviewHeight	
pixelAspectNum	New in Premiere Pro 2.0. The numerator and denominator of the pixel aspect ratio.
pixelAspectDen	

pmPlayTimebase

Selector: *playmod_NewList* (member of `pmNewListParms`)

Describes the timebase and duration of a clip.

```
typedef struct {
     csSDK_uint32    timeBase;
     csSDK_int32     samplesize;
     csSDK_int32     fileDuration;
} pmPlayTimebase;
```

`timeBase`	The timebase of the clip.
`samplesize`	The size of one sample.
`fileDuration`	The duration of the clip. This is equal to frames * samplesize

pmPutFrameRec

Selector: *playmod_PutFrame*

The host passes a rendered frame in. Display the frame.

```
typedef struct {
     csSDK_int32                size;
     csSDK_int32                version;
     PPixHand                   theFrame;
     pmPutFrameDestination      destination;
} pmPlayTimebase;
```

`size`	The size of the structure.
`version`	Version of the player API.
`theFrame`	The rendered frame to display.
`destination`	Location to display frame. Either `pmPutFrameDestination_VideoDesktop` or `pmPutFrameDestination_VideoHardware`.

pmPutFrameRequestRec

Selector: *playmod_PutFrameRequest*

The attributes of the frame the player would like to receive in a subsequent call to *playmod_PutFrame*.

```
typedef struct {
     csSDK_int32              size;
     csSDK_int32              version;
     pmPutFrameDestination    destination;
     csSDK_int32              width;
     csSDK_int32              height;
     PrPixelFormat            pixelFormats[64];
     csSDK_int32              pixelFormatCount;
     PrRenderQuality          quality;
} pmPlayTimebase;
```

size	The size of the structure.
version	Version of the player API.
destination	Location to display frame. Either pmPutFrameDestination_VideoDesktop or pmPutFrameDestination_VideoHardware.
width	The player's preferred frame width.
height	The player's preferred frame height.
pixelFormats[64]	Array of pixel format prefs.
pixelFormatCount	Actual number of pixel formats in array.
quality	Render quality flag.

pmPutTemporaryTimelineRec

Selector: *playmod_PutTemporaryTimeline*

Gives the player access to the timeline and position to display. timelineData is valid for a short period of time, ending on the next *playmod_Update* call.

```
typedef struct
{
     csSDK_int32    size;
     csSDK_int32    version;
     PrTimelineID   timelineData;
     PrTime         position;
} pmPutTemporaryTimelineRec;
```

size	The size of the structure.
version	Version of the player API.
timelineData	An identifier to be passed back to calls in the Timeline Functions.
position	The time to display.

pmStartupRec

Selector: *playmod_Startup*

Provide basic information on the player.

```
typedef struct {
     prPluginID     outPlayerID;
     prUTF16Char    outDisplayName[256];
} pmStartupRec;
```

`outPlayerID`	Provide the plug-in specified player ID. For the player to be used in an editing mode, this ID must match the `<EditingMode.Player>` element in an editing mode XML file.
`outDisplayName`	New in CS4. Provide the localized display name of the player.

PrVideoDisplayParameters

Selector: *playmod_SetVideoDisplayType*

The host specifies the display parameters for the current player.

```
typedef struct {
     PrVideoDisplayType       displayType;
     PrScopeDisplayIntensity  scopeIntensity;
     prBool                   scopeUseSetup;
     prBool                   scopeUseIRE;
     prBool                   scopeShowChroma;
     prBool                   scopeMagnify;
     csSDK_int32              reserved[23];
} PrVideoDisplayParameters;
```

displayType	One of the following types: kPrVideoDisplayType_Vectorscope, kPrVideoDisplayType_Waveform, kPrVideoDisplayType_RGBParade, kPrVideoDisplayType_YUVParade, kPrVideoDisplayType_VectWaveYParade, kPrVideoDisplayType_VectWaveRParade, kPrVideoDisplayType_Alpha, kPrVideoDisplayType_All, kPrVideoDisplayType_Off, kPrVideoDisplayType_DirectManipulation, kPrVideoDisplayType_Composite kPrVideoDisplayType_AudioOnly
scopeIntensity	A user-specified float value for the scope intensity, applicable to all scope view types.
scopeUseSetup	If non-zero, the user has checked the Setup checkbox, applicable to the YC Waveform view types.
scopeUseIRE	If non-zero, the scopes are using IRE units. Otherwise, they are using millivolts.
scopeShowChroma	If non-zero, the user has checked the Chroma checkbox, applicable to the YC Waveform view types.
scopeMagnify	Unused.

pmGetPosRec

Selector: *playmod_GetPos*

Tell the host the current position and play mode.

```
typedef struct {
     PrTime          position;
     pmPlayMode      mode;
} pmGetPosRec;
```

position	The current time position.
mode	The current mode. Either playmode_Stopped or playmode_Playing

pmPlayParms

Selector: *playmod_Preroll* and *playmod_Play*

The host specifies how and where to start and stop playing.

```
typedef struct {
     PrTime               inTime;
     PrTime               outTime;
     PrTime               startTime;
     prBool               loop;
     float                speed;
     prBool               export;
     PlayModuleDeviceID   deviceID;
     prBool               audioRecord;
} pmPlayParms;
```

`inTime`	In point. This is not necessarily the `startTime`, and is used when looping or playing backwards. If looping and playing backwards, jump to `outTime` when the movie reaches this frame, otherwise stop playing.
`outTime`	Out point. If looping, jump to `inTime` when the movie reaches this frame, otherwise stop playing.
`startTime`	Start point. The frame from which to start playing.
`loop`	If true, loop playback until you receive *playmod_Stop*.
`speed`	Playback speed. The speed value is a float, with 1.0 meaning forward play at normal speed and (-1.0) meaning backward play at normal speed.
`export`	If true, export to tape.
`deviceID`	If export is true, this ID will be non-zero to allow use of the Playmod Device Control Suite.
`audioRecord`	If true, the user has initiated a playback with voiceover audio recording. If the player uses host-based audio (InitHostAudio), then no special action is required. If the player uses plug-in based audio, then voiceover recording is not supported, and it must return `playmod_CannotRecord` to the host.

pmStepRec

Selector: *playmod_Step*

The host specifies the next position step.

```
typedef struct {
     PrTime    stepDistance;
} pmStepRec;
```

stepDistance	How far to step. A positive value is a step forward, and a negative value is a step backward.

pmNewListParms

Selector: *playmod_NewList*

Create a new cutlist.

```
typedef struct {
     pmPlayTimebase      listTimebase;
     pmDisplayPos        unused;
     PrPlayID            playID;
     PrTimelineID        timelineData;
     PrAudioChannelType  audioChannelType;
     PrMemoryPtr         playmodPrefs;
     int                 width;
     int                 height;
     csSDK_uint32        pixelAspectNum;
     csSDK_uint32        pixelAspectDen;
     prFieldType         fieldType;
     float               audioSampleRate;
     csSDK_uint32        videoSubType;
} pmNewListParms;
```

listTimebase	The timebase for the new cutlist.
playID	ID of this play instance, used for callback functions.
timelineData	An identifier to be passed back to calls in the Timeline Functions.
audioChannelType	The audio channel type.
playmodPrefs	A pointer to the player preference data.
width	New in Premiere Pro 2.0. The dimensions.
height	
pixelAspectNum	New in Premiere Pro 2.0. The numerator and denominator of the pixel aspect ratio.
pixelAspectDen	
fieldType	New in Premiere Pro CS3. Field dominance. One of either prFieldsNone, prFieldsUpperFirst, prFieldsLowerFirst, or prFieldsUnknown.
audioSampleRate	New in Premiere Pro CS3.
videoSubType	New in Premiere Pro CS3.

prtPlayableRangeRec

Selector: *playmod_GetRTStatusForTime*

Note: This structure has changed in CS4. Defines the real-time ability of a segment. If the segment at the `position` is an empty area in the timeline that continues to the end, the player should return the `PRT_END_OF_TIMELINE` value in the `outEndTime` field.

```
typedef struct {
     csSDK_int32    size;
     csSDK_int32    version;
     PrTime         position;
     PrTime         outEndTime;
     prtPlaycode    playcode;
} prtPlayableRangeRec;
```

`size`	Size of the entire structure.
`version`	Version of the structure, now `PRT_VERSION_PLAYABLERANGEREC_PREM10`. Version before CS4 was `PRT_VERSION_PLAYABLERANGEREC_PREM7`.
`position`	The frame the host is asking the real-time ability for.
`outEndTime`	Provide the out time of the segment at `position`.
`playcode`	Provide the appropriate code for the real-time ability of the segment: `PRT_PLAYCODE_REALTIME` - no rendering required `PRT_PLAYCODE_NON_REALTIME_UNSPECIFIED` - red bar when unrendered, green bar when rendered `PRT_PLAYCODE_REALTIME_WITH_MISMATCH` - new in CS4, if a clip can playback realtime, but has a media mismatch with the project, show that clip as yellow. When the user renders the work area, the yellow segments will not be rendered and will remain yellow. There is a new sequence command, Render All, that will render the yellow segments in addition to the red segments. On Export To Tape, the yellow segments will be rendered.

Suites

For information on how to acquire and manage suites, see the SweetPea Suites section.

Playmod Audio Suite

This suite is used to play audio during both playback and scrubbing.

Host-Based, or Plug-in Based Audio?

A player has two choices for playing audio: it can ask the host to play the audio through the audio device selected by the user, or it can get audio buffers from the host and handle its own playback of audio. It's possible to use host audio and plug-in audio at different times in the same player. For example, in the built-in DV editing mode, the built-in player will use plug-in audio if playing out to the DV device. But if just playing to the Sequence Monitor in the application UI, it will use host audio instead.

Players that use plug-in based audio can provide details to be used in the Preferences > Audio Output Mapping dialog. They should use `InitPluginAudio` from version 2 of the Playmod Audio Suite, and implement `playmod_GetAudioInfo` and `playmod_GetAudioChannelInfo`.

Audio Playback

For host-based audio, the following sequence of calls should happen:
- Plugin receives *playmod_Preroll*.
- Plugin calls `initHostAudio`.
- Plugin returns from *playmod_Preroll*.
- Plugin receives *playmod_Play*.
- Plugin calls `startAudio`.
- Plugin plays.
- Plugin stops playback.
- Plugin calls `stopAudio`.

During playback, the plugin can get the current play position from the host using `getCurrentTime`, which will be clocked from the audio device.

For plugin-based audio, the following sequence should happen:
- Plugin receives *playmod_Preroll*.
- Plugin calls `initPluginAudio`, passing a `AudioPlaybackSettings` struct holding the current audio playback settings.
- Plugin returns from *playmod_Preroll*.
- Plugin receives *playmod_Play*.
- Plugin calls `startAudio`.
- Plugin plays. Whenever the plugin needs another buffer of audio, it calls `getNextAudio-Buffer`. Note that these buffers are always sequential.
- Plugin stops playback.
- Plugin calls `stopAudio`.

Note that the `initAudio` and `startAudio` calls can fail by returning an error code. If this happens, then the player must abort preroll or playback start.

The host will return a negative countdown during its own audio preroll, which will last at least as long as the requested minimum preroll time in the `StartAudio` call.

Audio Scrubbing

The player will receive a *playmod_EnterScrub* selector. It should call `InitHostAudio` or `InitPluginAudio` at this point, passing in that it is in scrub mode, followed by a `StartAudio` call. If `InitHostAudio` was called, it should pass into `StartAudio` an `AudioPositions` structure with the `currentPosition`, `inPosition`, and `outPosition` all the same, so that no audio will be played.

After this, the player will receive one or more *playmod_SetPos* selector. When it gets such a selector, it should play the audio associated with that video frame. Call `SetRange`, with the bounds being the length of the video frame. For host-based audio, this range will immediately begin playing. For plug-in-based audio, the plug-in can now request the next buffer and it will return audio sample frames from the correct position. The host will output silence after the out point in `SetRange` is reached.

When the player receives *playmod_LeaveScrub*, it should call `StopAudio`.

The player will never receive a *playmod_Preroll* or *playmod_Play* call while in scrub mode.

AudioTimeCallback

This callback definition is for the host to tell the plugin what the current audio time is while it is playing the audio. It will be called on a very high priority thread and it is important that the plugin not block this thread or perform any lengthy processing during this call.

```
typedef void (*AudioTimeCallback)(
     void*     inInstanceData,
     PrTime    inCurrentTime);
```

InitHostAudio

Tell the host to initialize the audio hardware for the host to play the audio. `GetNextAudioBuffer` will be unavailable using this initialization. Returns either `suiteError_NoError` or `suiteError_PlayModuleAudioInitFailure`.

```
prSuiteError (*InitHostAudio)(
     csSDK_int32         inPlayID,
     AudioTimeCallback   inTimerCallback,
     void*               inInstanceData,
     int                 inIsScrubbing,
```

```
    PrTime*             outClockInterval);
```

Parameter	Description
`inTimerCallback`	This is an optional parameter. If NULL, then the player will not be called by the audio hardware thread. If non-NULL, it must point to a function which can handle being called on a high-priority thread at a regular interval.
`inInstanceData`	If `inTimerCallback` is non-NULL, then the player can pass a pointer to instance data that it wishes returned when the timer callback is called.
`inIsScrubbing`	If non-zero, the host will treat this as a scrubbing init.
`outClockInterval`	If `inTimerCallback` is non-NULL and this parameter is non-NULL, then on return it will contain the expected calling interval of `inTimerCallback`. This should be used as a guide only, and may vary depending on the audio hardware.

InitPluginAudio

Tell the host to initialize the audio render chain for the plugin to play the audio. A player that calls InitPluginAudio must also specify the audio hardware attributes during *playmod_GetAudioInfo*. GetNextAudioBuffer will be available using this initialization. Returns either `suiteError_NoError` or `suiteError_PlayModuleAudioInitFailure`.

```
prSuiteError (*InitPluginAudio)(
     csSDK_int32                    inPlayID,
     int                            inIsScrubbing,
     const AudioPlaybackSettings*   inSettings);
```

Parameter	Description
`inIsScrubbing`	If non-zero, the host will treat this as a scrubbing init.
`inSettings`	The settings for the requested playback.

StartAudio

Tell the host to start the audio playback. If `InitHostAudio` was used, then the playback clock will start running on another thread, possibly before this call returns. If `InitPluginAudio` was used, then `GetNextAudioBuffer` is now available. Returns `suiteError_NoError`, `suiteError_PlayModuleAudioNotInitialized`, or `suiteError_PlayModuleIllegalPlaySetting`.

```
prSuiteError (*StartAudio)(
     csSDK_int32               inPlayID,
     const AudioPositions*     inPlayPosition,
     float                     inPlaybackSpeed,
     PrTime                    inMinimumPreroll);
```

Parameter	Description
inPlayPosition	The playback position structure
inPlaybackSpeed	The requested playback speed. Must be non-zero
inMinimumPreroll	The minimum time that the player needs to preroll video. The host guarantees that the audio clock will start at least this long before the start point.

GetNextAudioBuffer

Retrieves from the host the next contiguous requested number of audio sample frames, specified in inNumSampleFrames, in inInBuffers as arrays of uninterleaved floats. The plug-in must manage the memory allocation of inInBuffers, which must point to n buffers of floating point values of length inNumSampleFrames, where n is the number of channels. This call is only available if InitPluginAudio was used. Returns suiteError_NoError, suiteError_PlayModuleAudioNotInitialized, or suiteError_PlayModuleAudioNotStarted.

```
prSuiteError (*GetNextAudioBuffer)(
     csSDK_int32    inPlayID,
     float**        inInBuffers,
     float**        outOutBuffers,
     unsigned int   inNumSampleFrames);
```

Parameter	Description
inInBuffers	A pointer to an array of buffers holding inNumSampleFrames input audio in each buffer, corresponding to the total number of available input channels.
outOutBuffers	A pointer to an array of buffers inNumSampleFrames long into which the host will write the output audio. There must be N buffers, where N is the number of output channels for the output channel type specified in InitPluginAudio.

inNumSampleFrames	The size of each of the buffers in the array in both inInBuffers and outOutBuffers.

SetPosition

Change the current audio playback position. If it is outside the in or out points, then the appropriate action will be taken to set the position. Returns suiteError_NoError, suiteError_PlayModuleAudioNotInitialized, or suiteError_PlayModuleAudioNotStarted.

```
prSuiteError (*SetPosition)(
     csSDK_int32    inPlayID,
     const PrTime*  inRequestedPosition,
     PrTime*        outActualPosition);
```

Parameter	Description
inRequestedPosition	Points to the requested playback position.
outActualPosition	On return, will contain the actual position after the change (can be NULL).

GetPosition

This function will return the time of the audio buffer currently being played by the audio hardware. It is only valid to call after an Init call and until StopAudio is called. Returns suiteError_NoError, suiteError_PlayModuleAudioNotInitialized, or suiteError_PlayModuleAudioNotStarted.

```
prSuiteError (*GetPosition)(
     csSDK_int32    inPlayID,
     PrTime*        outPosition);
```

Parameter	Description
outPosition	On return, the time of the audio buffer currently being played by the audio hardware.

SetRange

Change the current audio in and out points, and the looping state. If the current position field contains a valid position (i.e. it is between the in and out points), then the playback position will be moved to this position. If it contains an illegal value, then the playback position will not be

moved except as constrained by the changed in and out points. So, to simply change the in and out points or looping state, here is an example of the `AudioPositions` struct on calling:

```
AudioPositions position(0, 1000000, -1, TRUE);
```

On return, the `currentPosition` value will be filled in with the current position. If instead the following struct is used:

```
AudioPositions position(0, 1000000, 25000, TRUE);
```

then the current position will be moved to 25000. Returns `suiteError_NoError`, `suiteError_PlayModuleAudioIllegalPlaySetting`, or `suiteError_PlayModuleAudioNotInitialized`.

```
prSuiteError (*SetRange)(
     csSDK_int32               inPlayID,
     const AudioPositions*     inRequestedPosition,
     AudioPositions*           outActualPosition);
```

Parameter	Description
`inRequestedPosition`	The requested state for the in, out, and loop positions, and, if valid, current position.
`outActualPosition`	On return, this will be the actual position that will be used even if an invalid position was sent in. Can be NULL.

SetPlaybackSpeed

Set the audio playback speed. This value must be non-zero, but can be any positive or negative value. Once the magnitude reaches a certain point, the host will generate silence. For plug-in-based audio, this call should be used to enable the host's audio resampling. For assurance, you can use the `GetPosition` call to ensure that the host stays in sync. If the plug-in does its own resampling, leaving this at 1. Note that you can never request a larger buffer from `GetNextAudioBuffer` than you initially declared in `InitPluginAudio`, so if the plug-in does its own resampling, at faster speeds it will need to break up the buffers into sections. Returns `suiteError_NoError`, `suiteError_PlayModuleAudioIllegalPlaySetting`, or `suiteError_PlayModuleAudioNotInitialized`.

```
prSuiteError (*SetPlaybackSpeed)(
     csSDK_int32    inPlayID,
     float          inSpeed);
```

Parameter	Description

inSpeed	The requested playback speed. Must be non-zero

StopAudio

StopAudio will cease audio playback and deinitialize the audio system. After calling StopAudio, you must call one of the InitAudio routines before calling StartAudio again. Returns suiteError_NoError, suiteError_PlayModuleAudioNotInitialized, or suiteError_PlayModuleAudioNotStarted.

```
prSuiteError (*StopAudio)(
     csSDK_int32    inPlayID);
```

AudioPlaybackSettings

This structure is passed in to the call to InitPluginAudio.

```
typedef struct AudioPlaybackSettings
{
     float                sampleRate;
     csSDK_uint32         maxBufferSize;
     PrTime               inputLatency;
     PrTime               outputLatency;
     csSDK_uint32         outNumInputChannels;
     csSDK_uint32         outNumOutputChannels;
} AudioPlaybackSettings2;
```

Member	Description
sampleRate	The sample rate the player requests audio at
maxBufferSize	The largest buffer (in sample frames) that the player will ever ask for.
inputLatency	The estimated delay from the actual sound to when the plug-in receives the audio. A sample latency calculation is: [buffer size] + [hardware input latency].
outputLatency	The estimated delay from when the plug-in receives the audio to when it is sent through the output hardware. A sample latency calculation is [buffer size] + [hardware output latency].
outNumInputChannels	Number of input channels. Passed back from InitPluginAudio.
outNumOutputChannels	Number of output channels. Passed back from InitPluginAudio.

AudioPositions

```
typedef struct AudioPositions
{
     PrTime    inPosition;
     PrTime    outPosition;
     PrTime    currentPosition;
     prBool    looping;
} AudioPositions;
```

Member	Description
`inPosition`	The in point of the playback. The in point is not necessarily the start point. For example, if playback begins in the middle of a sequence, then the start point will be after the in point. The in point must be less than or equal to the playback start and out points.
`outPosition`	The out point of playback. Must be greater than or equal to the playback start and in points.
`currentPosition`	The current playback time. Must be between the in and out points.
`looping`	If `kPrTrue`, then playback should loop at the boundaries.

Playmod Device Control Suite

This suite is used by players to control a hardware device during Export To Tape. The necessary `PlayModuleDeviceID` will only be passed down when a transmit is requested. The player must then call all of the methods in this suite, in the order in which they are listed. This suite includes the following callbacks and type:

Seek

Tells the device to seek to the appropriate location. Returns `kPrDeviceControlResult_Success`, `kPrDeviceControlResult_IllegalCallSequence`, or `kPrDeviceControlResult_GeneralError`.

```
prSuiteError (*Seek)(
     PlayModuleDeviceID inDeviceID);
```

Arm

Tells the device to prepare to record. Returns `kPrDeviceControlResult_Success`, `kPrDeviceControlResult_IllegalCallSequence`, or `kPrDeviceControlResult_GeneralError`.

```
prSuiteError (*Arm)(
     PlayModuleDeviceID inDeviceID);
```

Record

Tells the device to start recording. Returns `kPrDeviceControlResult_Success`, `kPrDeviceControlResult_IllegalCallSequence`, or `kPrDeviceControlResult_GeneralError`.

```
prSuiteError (*Record)(
     PlayModuleDeviceID inDeviceID);
```

Stop

Tells the device to stop recording. Returns `kPrDeviceControlResult_Success`, `kPrDeviceControlResult_IllegalCallSequence`, or `kPrDeviceControlResult_GeneralError`.

```
prSuiteError (*Stop)(
     PlayModuleDeviceID inDeviceID);
```

PlayModuleDeviceID

of type `csSDK_int32`

Playmod Render Suite

Get rendered video from the host. For best performance, use the asynchronous render requests with the source media prefetching calls, although synchronous rendering is available too. New in CS5, `GetAcceleratedRendererRTStatusForTime` provides a way for a player to find out if a segment uses effects accelerated by the Mercury Playback Engine or any other accelerated renderer.

PrRenderCacheType

Value	Description
kRenderCacheType_None	Don't cache any type of frames.
kRenderCacheType_ImportedFrames	Cache imported frames.
kRenderCacheType_ImportedStillFrames	Cache imported stills.
kRenderCacheType_IntermediateFrames	Cache intermediate frames.
kRenderCacheType_RenderedFrame	Cache rendered frames.
kRenderCacheType_AllFrames	Cache all frames.

PrSDKPlayModuleRenderSuite_AsyncCompletionProc

This plug-in specified function will be called by the host when asynchronous renders are completed. It is passed to the host to call in `SetRenderCompletionProc`.

```
void (*PrSDKPlayModuleRenderSuite_AsyncCompletionProc)(
     void*          inAsyncCompletionData,
     csSDK_int32    inRequestID,
     PPixHand       inPPixHand);
```

RenderVideoFrame

Renders a video frame synchronously. It will not return until the requested video frame has been rendered or an error has occurred.

```
prSuiteError (*RenderVideoFrame)(
     csSDK_int32               inPlayID,
     const PrTime*             inFrameTime,
     PPixHand*                 outRenderedFrame,
     const PrPixelFormat*      inRequestedPixelFormatArray,
     csSDK_int32               inRequestedPixelFormatArrayCount,
     const prRect*             inFrameRect,
     csSDK_uint32              inPixelAspectRatioNumerator,
     csSDK_uint32              inPixelAspectRatioDenominator,
     PrRenderQuality           inRenderQuality,
     PrRenderCacheType         inCacheFlags,
     prBool                    inRenderFields);
```

Parameter	Description
outRenderedFrame	The output PPix, the caller is responsible to destroy this

inFrameTime	The time of the video frame
inRequestedPixelFormatArray	An array of PF_PixelFormats that list your format preferences in order. This list must end with PrPixelFormat_BGRA_4444_8u
inRequestedPixelFormatArrayCount	Number of formats in the format array
inFrameRect	Video frame size
inPixelAspectRatioNumerator	The numerator for the pixel aspect ratio
inPixelAspectRatioDenominator	The denominator for the pixel aspect ratio.
inRenderQuality	The render quality of this frame.
inCacheFlags	The type of rendered frames to cache.
inRenderFields	Render fields

QueueAsyncVideoFrameRender

This will queue a render of a video frame. It will return immediately. If the frame was available in the cache, it will be returned from this call and the completion proc will not be called. If the frame is not available, the request will be queued and the completion proc will be called when the request is complete. Note: while inside your completion proc, no other video frames will be rendered so do not do any time consuming work in your completion proc. The async render completion proc must be set before calling this function.

```
prSuiteError (*QueueAsyncVideoFrameRender)(
     csSDK_int32                inPlayID,
     const PrTime*              inFrameTime,
     void*                      inAsyncCompletionData,
     csSDK_int32*               outRequestID,
     PPixHand*                  outRenderedFrame,
     const PrPixelFormat*       inRequestedPixelFormatArray,
     csSDK_int32                inRequestedPixelFormatArrayCount,
     const prRect*              inFrameRect,
     csSDK_uint32               inPixelAspectRatioNumerator,
     csSDK_uint32               inPixelAspectRatioDenominator,
     PrRenderQuality            inRenderQuality,
     prBool                     inRenderFields
     PrRenderCacheType          inCacheFlags);
```

Parameter	Description
inAsyncCompletionData	User-specific data sent to the async completion proc
inFrameTime	The time of the video frame

outRequestID	The request ID, if the frame is rendered async
outRenderedFrame	The output PPix, the caller is responsible to destroy this
inRequestedPixelFormatArray	An array of PF_PixelFormats that list your format preferences in order. This list must end with PrPixelFormat_BGRA_4444_8u
inRequestedPixelFormatArray-Count	Number of formats in the format array
inFrameRect	Video frame size
inPixelAspectRatioNumerator	The numerator for the pixel aspect ratio
inPixelAspectRatioDenominator	The denominator for the pixel aspect ratio.
inRenderQuality	The render quality of this frame.
inCacheFlags	The type of rendered frames to cache.
inRenderFields	Render fields

SetAsyncRenderCompletionProc

Sets the completion function that will be called when an async render request is completed. Note: while inside your completion proc, no other video frames will be rendered so do not do any time consuming work in your completion proc. Instance data is sent into each async render call, and will be returned to this completion proc.

```
prSuiteError (*SetAsyncRenderCompletionProc)(
     csSDK_int32                                inPlayID,
     PrSDKPlayModuleRenderSuite_AsyncCompletionProc inCompletionProc);
```

Parameter	Description
inPlayID	
inCompletionProc	This function will be called when async renders are completed

CancelOneOutstandingAsyncRequest

Cancel a specific render request for this RenderFrame ref.

```
prSuiteError (*CancelOneOutstandingAsyncRequest)(
     csSDK_int32    inPlayID,
     csSDK_int32    inAsyncRequestID);
```

CancelAllOutstandingAsyncRequests

Cancel all pending render requests for this RenderFrame ref.

```
prSuiteError (*CancelAllOutstandingAsyncRequests)(
     csSDK_int32    inPlayID);
```

FetchRenderedFrameFromCache

Returns the video frame if it is already in the cache. This does not cause a render to occur.

```
prSuiteError (*FetchRenderedFrameFromCache)(
     csSDK_int32               inPlayID,
     const PrTime*             inFrame,
     PPixHand*                 outRenderedFrame,
     const PrPixelFormat*      inRequestedPixelFormatArray,
     csSDK_int32               inRequestedPixelFormatArrayCount,
     const prRect*             inFrameRect,
     csSDK_uint32              inPixelAspectRatioNumerator,
     csSDK_uint32              inPixelAspectRatioDenominator,
     PrRenderQuality           inRenderQuality);
```

Parameter	Description
`inFrame`	The time of the video frame
`outRenderedFrame`	The output `PPix`, the caller is responsible to destroy this
`inRequestedPixelFormatArray`	An array of `PF_PixelFormats` that list your format preferences in order. This list must end with `PrPixelFormat_BGRA_4444_8u`
`inRequestedPixelFormatArrayCount`	Number of formats in the format array
`inFrameRect`	Video frame size
`inPixelAspectRatioNumerator`	The numerator for the pixel aspect ratio
`inPixelAspectRatioDenominator`	The denominator for the pixel aspect ratio.
`inRenderQuality`	The render quality of this frame.

PrefetchMedia

Prefetches the media needed to render this frame. This is a hint to the importers to begin reading media needed to render this video frame.

```
prSuiteError (*PrefetchMedia)(
     csSDK_int32    inPlayID,
     const PrTime*  inFrame);
```

PrefetchMediaWithRenderParameters

Prefetches the media needed to render this frame, using all of the parameters used to render the frame. This is a hint to the importers to begin reading media needed to render this video frame.

```
prSuiteError (*PrefetchMediaWithRenderParameters)(
     csSDK_int32              inPlayID,
     const PrTime*            inFrameTime,
     const PrPixelFormat*     inRequestedPixelFormatArray,
     csSDK_int32              inRequestedPixelFormatArrayCount,
     const prRect*            inFrameRect,
     csSDK_uint32             inPixelAspectRatioNumerator,
     csSDK_uint32             inPixelAspectRatioDenominator,
     PrRenderQuality          inRenderQuality,
     prBool                   inRenderFields);
```

CancelPrefetchMediaWithRenderParameters

New in CS4. Cancel a specific prefetch if playback has already passed the time for which the prefetch was needed.

```
prSuiteError (*CancelPrefetchMediaWithRenderParameters)(
     csSDK_int32              inPlayID,
     const PrTime*            inFrameTime,
     const PrPixelFormat*     inRequestedPixelFormatArray,
     csSDK_int32              inRequestedPixelFormatArrayCount,
     const prRect*            inFrameRect,
     csSDK_uint32             inPixelAspectRatioNumerator,
     csSDK_uint32             inPixelAspectRatioDenominator,
     PrRenderQuality          inRenderQuality,
     prBool                   inRenderFields);
```

CancelAllOutstandingMediaPrefetches

This will cancel all media pre-fetches that are still outstanding.

```
prSuiteError (*CancelAllOutstandingMediaPrefetches)(
     csSDK_int32    inPlayID);
```

AddFrameToCache

This will add a video frame to the cache. As an example, this can be used to add a video frame of another pixel format back to the cache. Then future requests for this video frame may be satisfied in this pixel format.

```
prSuiteError (*AddFrameToCache)(
     csSDK_int32    inPlayID,
     PPixHand       inOriginalPPix,
     PPixHand       inNewPPix);
```

Parameter	Description
`inPlayID`	
`inOriginalPPix`	The original `PPix` that was rendered
`inNewPPix`	The new `PPix` that also represents this frame

AllowTransparentVideoFrames

Set the type of video frame returned. If set to TRUE the video frame returned may have transparent alpha. Note that this type of video frame still needs to be composited on black. This is an option if you can perform the final composite in hardware, otherwise set to FALSE and the final composite onto black will be performed during the render.

```
prSuiteError (*AllowTransparentVideoFrames)(
     csSDK_int32    inPlayID,
     prBool         inAllowTransparentVideoFrames);
```

Parameter	Description
`inPlayID`	
`inAllowTransparentVideoFrames`	If `kPrTrue`, the resulting video frame will not be composited onto black

RefreshRTStatus

New in version 3. Force a refresh of the RT segment status for a given player.

```
prSuiteError (*RefreshRTStatus)(
     csSDK_int32    inPlayID);
```

GetAcceleratedRendererRTStatusForTime

New in version 4. Query the real-time status of an accelerated renderer if used. This is important for players to show the correct real-time status for a segment with effects accelerated using the Mercury Playback Engine, or any other accelerated renderer.

```
prSuiteError (*GetAcceleratedRendererRTStatusForTime)(
     csSDK_int32              inPlayID,
     prtPlayableRangePtr      outplayableRange);
```

Scope Render Suite

A player receives enough information to display its own scopes. It can optionally use the Scope Render Suite to have the host to render the scopes. The Scope Render Suite processes frame buffers of either `BGRA_4444_8u` or `VUYA_4444_8u`. It does not support other pixel formats.

In version 2 of this suite, RenderScope now renders to a `prWnd` rather than a `prOffscreen`.

Stock Image Suite

New in CS3. Provides four images useful for players: color bars, not yet rendered, playing on hardware, and capture preview on hardware. Starting in CS4, two more images are now retrievable from the suite: media pending and media offline.

9 Transitions

Transitions take two source frames and process them into a single destination frame. Transitions can have their own custom modal setup dialog for additional settings.

Getting Started

Begin with the sample project, progressively replacing its functionality with your own.

Resources

More than other plug-in types, transitions depend heavily on the PiPL. After making changes to the PiPL, rebuild the plug-in each time, so that the PiPL will be recompiled.

A Transition PiPL Example

```
#include "PrSDKPiPLVer.h"
#ifndef PRWIN_ENV
#include "PrSDKPiPL.r"
#endif

// The following three strings should be localized
#define plugInName "Cool Video Transition"
#define description "Image A warps into image B."
#define plugInCategory "SDK Transitions"

// This name should not be localized or updated
#define plugInMatchName "SDK Cool Transition"

resource 'PiPL' (16000) {
{
      // The plug-in type
```

```
      Kind {PrEffect},

      // The plug-in name as it will appear to the user
      Name {plugInName},

      // The internal name of this plug-in
      AE_Effect_Match_Name {plugInMatchName},

      // The folder containing the plug-in in the Effects Panel
      Category {plugInCategory},

      // The version of the PiPL resource definition
      AE_PiPL_Version {PiPLVerMajor, PiPLVerMinor},

      // Transition effect description, this should be localized
      Pr_Effect_Description {description},

      // Transition effect info flags
      Pr_Effect_Info {
#if (PiPLVerMinor == 2) & (PiPLVerMinor > 1)
            1,                              // 1 - Version Number
#endif
            bitNone,                        // 2 - Valid Corner Bits
            bitNone,                        // 3 - Initial Corner Bits
#if (PiPLVerMinor == 2) & (PiPLVerMinor > 1)
            false,                          // unused
            noExclusiveDialog,              // 4 - Exclusive Dialog
            doesNotNeedCallbacksAtSetup,    // 5 - Callbacks at setup
            noDirectCompData,               // 6 - Internal use only
#else
            doesNotNeedCallbacksAtSetup,    // 5 - Callbacks at setup
            false,                          // unused
#endif
            dontWantInitialSetupCall,       // 7 - Initial setup call
            treatAsTransition,              // 8 - 2 Input Filter
            noCustomDialog,                 // 9 - Custom Dialog
            dontHighlightOppositeCorners,   // 10 - Highlight Opposite Corners
            notExclusive,                   // 11 - Exclusive
            notReversible,                  // 12 - Reversible
            doesNotHaveEdges,               // 13 - Has Edges
            doesNotHaveStartPoint,          // 14 - Start Point
            doesNotHaveEndPoint             // 15 - End Point
      }
}
};
```

Resources Table

Line	Name	Use	Valid States
1	Version Number	Version number of this property	Set to 1
2	Valid Corner Bits	Describes all the valid corner bits	`bitNone, bitTop, bitRight, bitBottom, bitLeft, bitUpperRight, bitLowerRight, bitLowerLeft, bitUpperLeft`
3	Initial Corner Bits	Describes all the initial corners displayed in the default setup	Same as Valid Corner Bits
4	Exclusive Dialog	Set to add a Setup button to the Effect Controls Panel. When the user hits the button, *esSetup* will be sent.	`noExclusiveDialog, needsExclusiveDialog`
5	Callbacks at Setup	Set if the transition uses previous frames; otherwise, the callback pointer is invalid during *esSetup*	`doesNotNeedCallbacksAtSetup, needsCallbacksAtSetup`
6	Internal Use only	[TODO] No effect?	`noDirectCompData`
7	Initial Setup Call	Set if you want to receive esSetup when your transition is first applied	`wantInitialSetupCall, dontWantInitialSetupCall`
8	Two Input Filter	Set if your transition is time-invariant (doesn't require interpolation); i.e., is really a two-input filter, not a transition	`treatAsTransition, treatAsTwoInputFilter`
9	Custom Dialog	Set to add a Setup button to the Effect Controls Panel. When the user hits the button, esSetup will be sent.	`noCustomDialog, hasCustomDialog`
10	Highlight Opposite Corners	Set if opposite corners bits are always to be highlighted simultaneously	`dontHighlightOppositeCorners, highlightOppositeCorners`
11	Exclusive	If set, the corner arrows will act like radio buttons, otherwise they act like checkboxes.	`notExclusive, exclusive`

12	Reversible	Set if the transition is reversible (can proceed either from source 1 to source 2); a direction control will be shown.	`notReversible, reversible`
13	Has Edges	If set, the anti-aliasing level control, the border thickness slider, and the border color controls will be shown.	`doesNotHaveEdges, haveEdges`
14	Start Point	If set, the transition supports a movable start point; a start point will appear in the Start thumbnail in the Effect Controls Panel.	`doesNotHaveStartPoint, haveStartPoint`
15	End Point	If set, the transition supports a movable end point; an end point will appear in the End thumbnail in the Effect Controls Panel	`doesNotHaveEndPoint, haveEndPoint`

Entry Point

```
short xEffect (
     short           selector,
     EffectHandle    theData)
```

selector is the action Premiere wants the transition to perform. `EffectHandle` provides source and destination buffers, and other useful information. Return `esNoErr` if successful, or an appropriate return code.

Selector Table

This table summarizes the various selector commands a transition can receive.

Selector	Description
esSetup	(optional) Allocate memory for your parameters if necessary. Display your modal setup dialog with default parameter values or previously stored values. Save the new values to `specsHandle`.
esExecute	Execute the transition using the stored parameters from `specsHandle`. Be aware of interlaced video, and don't overlook the alpha channel!
esDisposeData	(optional) Dispose of any instance data created during *esExecute*.

esCanHandlePAR	(optional) Tell Premiere how your transition handles pixel aspect ratio.
esGetPixelFormatsSupported	(optional) Return the pixel formats supported. Called iteratively until all formats have been given.
esCacheOnLoad	(optional) Return `esDoNotCacheOnLoad` to disable plug-in caching for this transition.

Selector Descriptions

esSetup

Optional. Display your custom settings dialog, populated with initial values from data stored in `specsHandle`. If it is NULL, show your dialog with default values and store results in a properly-sized handle in `specsHandle`. *esSetup* is only sent if you set `hasCustomDialog` in the `Pr_Effect_Info` property of the PiPL resource. Store any updated values back to `specsHandle` when done.

During *esSetup*, the frames passed to `EffectRecord.source1` and `source2` are always 320x240. The frame is the layer the transition is applied to at the current time indicator. If the CTI is not on the clip the transition is applied to, the frame is transparent black. If the transition has a setup dialog, the `FXCallbackProcPtr` should be used to get source frames for previews. `getPreviewFrameEx` can be used to get rendered frames, although if this call is used, the video filter should be ready to be called reentrantly with *fsExecute*.

esExecute

This is really the only required selector for a transition, and it's where the rendering happens. Take the input frames in `EffectRecord.source1` and `EffectRecord.source2`, render the transition and return the frame to Premiere in `EffectRecord.destination`. The `specsHandle` contains your parameter settings (already interpolated). You can store a handle to any additional non-parameter data in `EffectRecord.InstanceData`. If you do so, respond to *esDisposeData*, or your plug-in will leak memory.

The video your transition receives may be interlaced, in the field order determined by the sequence settings. Respect the rowbytes. Transitions must preserve the alpha channel, so that they will be correctly composited with underlying clips. Transitions may also have only an incoming clip, or only an outgoing clip, in which case one of the sources will be transparent alpha. Transitions should never assume the part and total values are related to frames. They should just always render based on the percent of part divided by total.

esDisposeData

Optional. Dispose of any instance data you allocated using Premiere's memory callbacks during *esExecute*. See `EffectRecord.InstanceData`.

esCanHandlePAR

Optional. Indicate how your transition wants to handle pixel aspect ratio by returning a combination of the following flags.

This selector is only sent if several conditions are met. The pixel aspect ratio of the clip to which the transition is applied must be known, and not be square (1.0). The clip must not be a solid color. The PiPL bits `anyPixelAspectRatio` and `unityPixelAspectRatio` must not be set. Note that even if an AspectRatio bit appears in the PiPL, it will not be set if the PiPL version is less than 2.3.

Flag	Description
`prEffectCanHandlePAR`	Premiere should not make any adjustment to the source image to compensate for PAR
`prEffectUnityPARSetup`	Premiere should render the source image to square pixels during *esSetup*
`prEffectUnityPARExecute`	Premiere should render the source image to square pixels during *esExecute*

esGetPixelFormatsSupported

Optional. Return the pixel formats supported. Called iteratively until all formats have been given. Set `EffectRecord.pixelFormatSupported` to a supported pixel format, and return `esNoErr`. When all formats have been described, return `esBadFormatIndex`. See the transition sample for an example.

esCacheOnLoad

Optional. Return `esDoNotCacheOnLoad` to disable plug-in caching for this transition.

Return Codes

Return Code	Reason
`esNoErr`	Operation has completed without error.

esBadFormatIndex	Return from *esGetPixelFormatsSupported* when all pixel formats have been enumerated.
esDoNotCacheOnLoad	Return from *esCacheOnLoad* to disable plug-in caching for this transition.
esUnsupported	The selector is not recognized, or unsupported.

EffectRecord

A transition is passed a handle to an `EffectRecord` with almost every selector.

```
typedef struct {
     PrMemoryHandle      specsHandle;
     PPixHand            source1;
     PPixHand            source2;
     PPixHand            destination;
     csSDK_int32         part;
     csSDK_int32         total;
     char                previewing;
     unsigned char       arrowFlags;
     char                reverse;
     char                source;
     prPoint             start;
     prPoint             end;
     prPoint             center;
     void *              privateData;
     FXCallBackProcPtr   callBack;
     BottleRec *         bottleNecks;
     short               version;
     short               sizeFlags;
     csSDK_int32         flags;
     TDB_TimeRecord *    tdb;
     piSuitesPtr         piSuites;
     PrTimelineID        timelineData;
     PrMemoryHandle      instanceData;
     char                altName[MAX_FXALIAS];
     PrPixelFormat       pixelFormatSupported;
     csSDK_int32         pixelFormatIndex;
     csSDK_uint32        instanceID;
     TDB_TimeRecord      tdbTimelineLocation;
     csSDK_int32         sessionPluginID;
} EffectRecord, **EffectHandle;
```

specsHandle	Instance settings, persistent across Premiere sessions. Create this handle during *esSetup*. Only used by transitions with custom parameters; must be created using newHandle.
source1	PPixHand for the incoming (the clip on the left of the transition) source video frame.
source2	PPixHand for the outgoing (the clip on the right of the transition) source video frame.
destination	PPixHand for the destination video frame, always the same size as source1 and source2. Store the output frame here during *esExecute*.
part	How far into the transition you are. part varies from 0 to total, inclusive.
total	Total length of the transition. Divide part by total to calculate the percentage of the transition to perform for a given *esExecute*. This value doesn't always correspond to frames or fields.
previewing	Unsupported
arrowFlags	Corner flags, set by the user.
reverse	If set, the frames are reversed.
source	[TODO] Is this still used?
start	Starting point of the transition (relative to the center point), specified by the user. Only meaningful if movablestartpoint is specified in the PiPL.
end	End point of the transition (relative to the center point). Only meaningful if movablestartpoint is specified in the PiPL.
center	Center point (for transitions that open and close).
privateData	Data private to Premiere. Pass to the frame-retrieval callback when requesting a frame.
callBack	Pointer to a callback for retrieving frames (or fields, for interlaced video) from source clips.
bottleNecks	Pointer to Premiere's bottleRec functions.
version	Version of this structure (kEffectVersion). Premiere Pro CS5 = TRANSITION_VERSION_11 Premiere Pro CS3 = TRANSITION_VERSION_10
sizeFlags	Field-rendering information.
flags	If doing a lower quality render, Premiere will pass in kEffectFlags_DraftQuality during *esExecute*. The transition can then optionally render a faster, lower-quality image for previewing.
tdb	Pointer to a time database record describing the project's timebase.

piSuites	Pointer to callback piSuites.
timelineData	Only available during *esSetup*. This opaque handle to the timeline database is required by timelineFuncs callbacks available in piSuites. This handle is useful in order to have a preview in a modal setup dialog during *esSetup*.
instanceData	Handle to private instance data that persists across invocations. Allocate the memory for this during *esExecute* and deallocate during *esDisposeData*. Do not use this field during *esSetup*.
altName	Unused.
pixelFormatSupported	Only valid during *esGetPixelFormatsSupported*. Return pixel format supported.
pixelFormatIndex	Only valid during *esGetPixelFormatsSupported*. Index of query of pixel type supported.
instanceID	The runtime instance ID uniquely identifies filters during a session. This is the same ID that is passed to players in prtTransitionRec.
tdbTimelineLocation	A time database record describing the location of the transition in sequence. Only valid during *esSetup*.
sessionPluginID	This ID should be used in the File Registration Suite for registering external files (such as textures, logos, etc) that are used by a plug-in instance but do not appear as footage in the Project Panel. Registered files will be taken into account when trimming or copying a project using the Project Manager.

FXCallBackProcPtr

Pointer to a callback for retrieving frames (or fields, for interlaced video) from source clips. Always available during esExecute, but only valid during esSetup if needsCallbacksAtSetup is set in the PiPL. Do not expect real-time performance from this callback.

```
typedef short (CALLBACK *FXCallBackProcPtr)(
     csSDK_int32    frame;
     short          track;
     PPixHand       thePort;
     prRect *       theBox;
     PrMemoryHandle privateData);
```

Parameter	Description
frame	Frame (or field, for interlaced video) requested.

`track`	Set this to 0 to get the incoming source (the clip on the left of the transition), and 1 to get the outgoing source (the clip on the right of the transition)
`thePort`	`PPixHand` where Premiere will store the frame
`theBox`	Bounds of the frame you want Premiere to retrieve.
`privateData`	Handle provided by Premiere in `VideoRecord.private-Data`

sizeFlags

For `sizeFlags`, the following bit flags are of interest:

Flag	Description
gvFieldsEven	The transition should render upper-field dominance
gvFieldsOdd	The transition should render lower-field dominance
gvFieldsFirst	The transition is currently rendering the dominant field

Additional Details

Fields and Field Processing

In an interlaced project, Premiere calls your transition once per field. This allows transitions such as wipes to have interlaced motion. `(*theData)->total` will be twice as large, each frame will be half-height, and rowbytes will double.

Respect the value of `rowbytes` when traversing data or the output will be incorrect.
Caching Behavior

Frame Caching

The rendered output of transitions is stored in the host media cache. For example, when the user scrubs over a frame with a transition on it, the transition will be called to render its effect on the frame and return the buffer to Premiere. Premiere caches the returned frame, so when the user scrubs over the same frame, Premiere will return the cached frame without having to call the transition again. If the user has modified the transition settings, the clip settings, the preview quality, etc, Premiere will call the transition to render with the new settings, but will keep the previously cached frame for a while. So if the changes are reversed, Premiere may still have the cached frame to return when appropriate.

Real-Time Transitions

In order to allow for real-time previews in transitions, an opaque handle to the current timelineData is passed to the transition during *esSetup* for use with `piSuites->timelineFuncs`.

Use this handle to obtain a real-time preview, provided by a player (provided you can communicate directly with the player; not every transition can be made real-time).

10 Video Filters

Video filters process a video frame into a destination frame. Filter parameters can vary with time. Premiere provides basic user interface in the Effect Controls panel, drawing sliders, color pickers, angle dials, and checkboxes based on the parameter definitions in the PiPL resource. Video filters can have their own custom modal setup dialog for additional settings.

We strongly recommend using the After Effects SDK to develop effects plug-ins. Almost all of the effects included in Premiere Pro are After Effects plug-ins.

If you've never developed a video filter before, you can skip the What's New section, and go directly to Getting Started.

What's New

What's New in Premiere Pro CS5?

In the Effects panel, video filters now appear with badges to advertise if they support YUV, 32-bit, and accelerated rendering. The user can filter the list of effects to show only the effects that support those rendering modes. Video filters will automatically receive YUV and 32-bit badges if they advertise support using the existing *fsGetPixelFormatsSupported*. Custom badges can also be created. See Effect Badging for more information.

What's New in Premiere Pro CS3?

Checkbox controls are now supported directly in the Effect Controls panel.

Filters can specify whether or not they want a setup button in the Effect Controls panel during *fsHasSetupDialog*, by returning `fsHasNoSetupDialog` or `fsNoErr`. Previously, this was set in the PiPL resource.

Getting Started

Begin with one of the two video filter sample projects, progressively replacing its functionality with your own.

Resources

Filter plug-ins can use PiPL resources to define their behaviors and supported properties. To provide any parameters in the Effect Controls panel, they must be defined in the PiPL in `ANIM_ParamAtom` sections, as demonstrated in the example below. The 'no UI' UI type is for non-key-frameable parameters. After making changes to the PiPL, rebuild the plug-in each time, so that the PiPL will be recompiled.

A Filter PiPL Example

```
#include "PrSDKPiPLVer.h"
#ifndef PRWIN_ENV
#include "PrSDKPiPL.r"
#endif

// The following two strings should be localized
#define plugInName "Cool Video Filter"
#define plugInCategory "SDK Filters"

// This name should not be localized or updated
#define plugInMatchName "SDK Cool Filter"

resource 'PiPL' (16000) {
{
      // The plug-in type
      Kind {PrEffect},

      // The plug-in name as it will appear to the user
      Name {plugInName},

      // The internal name of this plug-in
      AE_Effect_Match_Name {plugInMatchName},

      // The folder containing the plug-in in the Effects Panel
      Category {plugInCategory},

      // The version of the PiPL resource definition
      AE_PiPL_Version {PiPLVerMajor, PiPLVerMinor},
```

```
        // The ANIM properties describe the filter parameters, and also how the data is stored
                in the project file.  There is one ANIM_FilterInfo property followed by n ANIM_
                ParamAtoms
        ANIM_FilterInfo {
                0,
#ifdef PiPLVer2p3
                // Non-square pixel aspect ratio supported
                notUnityPixelAspectRatio,
                anyPixelAspectRatio,
                reserved4False,
                reserved3False,
                reserved2False,
#endif
                reserved1False, // These flags are for use by After Effects
                reserved0False, // Not used by Premiere
                driveMe, // Not used by Premiere
                needsDialog, // Not used by Premiere
                paramsNotPointer, // Not used by Premiere
                paramsNotHandle, // Not used by Premiere
                paramsNotMacHandle, // Not used by Premiere
                dialogNotInRender, // Not used by Premiere
                paramsNotInGlobals, // Not used by Premiere
                bgAnimatable, // Not used by Premiere
                fgAnimatable, // Not used by Premiere
                geometric, // Not used by Premiere
                noRandomness, // Not used by Premiere
                // Put the number of parameters here
                2,
                plugInMatchName
        },

        // There is one ANIM_ParamAtom for each parameter
        ANIM_ParamAtom {
                // This is the first property - Zero based count
                0,
                // The name to appear for the control
                "Level",
                // Parameter number goes here - One based count
                1,
                // Put the data type here
                ANIM_DT_SHORT,
                // UI control type
                ANIM_UI_SLIDER,
                0x0,
```

```
            0x0,  // valid_min (0.0)
            0x405fc000,
            0x0,  // valid_max (127.0)
            0x0,
            0x0,  // ui_min (0.0)
            0x40590000,
            0x0,  // ui_max (100.0)
#if PiPLVer2p3
            // New - Scale/dontScale UI Range if user modifies
            dontScaleUIRange,
#endif
            // Set/don't set this if the param should be animated
            animateParam,
            dontRestrictBounds,  // Not used by Premiere
            spaceIsAbsolute,  // Not used by Premiere
            resIndependent,  // Not used by Premiere
            // Bytes size of the param data
            2
      },

      ANIM_ParamAtom {
            1,
            "Target Color",
            2,
            // Put the data type here
            ANIM_DT_COLOR_RGB,
            // UI control type
            ANIM_UI_COLOR_RGB,
            0x0,
            0x0,
            0x0,
            0x0,
            0x0,
            0x0,
            0x0,
            0x0,
#ifdef PiPLVer2p3
            dontScaleUIRange,
#endif
            // Set/don't set this if the param should be animated
            animateParam,
            dontRestrictBounds,
            spaceIsAbsolute,
            resIndependent,
            // Bytes size of the param data
```

```
            4
      },
}
};
```

Entry Point

```
short xFilter (
      short           selector,
      VideoHandle     theData)
```

selector is the action Premiere wants the video filter to perform. `EffectHandle` provides source and destination buffers, and other useful information. Return `fsNoErr` if successful, or an appropriate return code.

Selector Table

This table summarizes the various selector commands a video filter can receive.

Selector	Description
fsInitSpec	(optional) Allocate and initialize your parameters with default values without popping a modal setup dialog.
fsHasSetupDialog	(optional) New for Premiere Pro CS3. Specify whether or not the filter has a setup dialog.
fsSetup	(optional) Allocate memory for your parameters if necessary. Display your modal setup dialog with default parameter values or previously stored values. Save the new values to `specsHandle`.
fsExecute	Filter the video using the stored parameters from `specsHandle`. Be aware of interlaced video, and don't overlook the alpha channel!
fsDisposeData	(optional) Dispose of any instance data created during *fsExecute*.
fsCanHandlePAR	(optional) Tell Premiere how your effect handles pixel aspect ratio.
fsGetPixelFormatsSupported	(optional) Gets pixel formats supported. Called iteratively until all formats have been given.
fsCacheOnLoad	(optional) Return `fsDoNotCacheOnLoad` to disable plug-in caching for this filter.

Selector Descriptions

fsInitSpec

Responding to this selector is optional. This selector is sent when the filter is applied to a clip and the plug-in is called for the first time. This call can be used to initialize the plug-in parameters with default values in order to achieve an initial "silent setup", in which *fsSetup* is skipped when the filter is applied to a clip, to avoid popping the modal dialog that may be needed in *fsSetup*.

Allocate and pass back a handle to a structure containing the parameter values in specsHandle. The filter is given the total duration (in samples), and number of the first sample in the source buffer.

fsHasSetupDialog

New for Premiere Pro CS3. Optional. Specify whether or not the filter has a setup dialog, by returning `fsHasNoSetupDialog` or `fsNoErr`.

fsSetup

Optional. Sent when the filter is applied, if *fsInitSpec* doesn't allocate a valid `specsHandle`. Also sent when the user clicks on the setup link in the Effect Controls Panel. The filter can optionally display a (platform-dependent) modal dialog to get new parameter values from the user. First, check `VideoHandle.specsHandle`. If NULL, the plug-in is being called for the first time. Initialize the parameters to their default values. If non-NULL, load the parameter values from `specsHandle`. Now use the parameter values to display a modal setup dialog to get new values. Return a handle to a structure containing the parameter values in `specsHandle`.

In order to properly store parameter values between calls to the plug-in, describe the structure of your `specsHandle` data in your PiPL's `ANIM` properties. Premiere interpolates animatable parameter values as appropriate before sending *fsExecute*.

The filter is given the total duration in samples and the sample number of the first sample in the source buffer. `VideoHandle.source` contains the first frame in the clip the filter is applied to, scaled to a 240 by 180 PPix, for preview purposes.

During *fsSetup*, the frames passed to `VideoRecord.source` are always 320x240. The frame is the layer the filter is applied to at the current time indicator. If the CTI is not on the clip the filter is applied to, the frame is transparent black. If the filter has a setup dialog, the `VFilterCallbackProcPtr` should be used to get source frames for previews. `getPre-`

viewFrameEx can be used to get rendered frames, although if this call is used, the video filter should be ready to be called reentrantly with *fsExecute*.

fsExecute

This is really the only required selector for a video filter, and it's where the rendering happens. Take the input frame in `VideoHandle.source`, render the effect and return the frame to Premiere in `VideoHandle.destination`. The `specsHandle` contains your parameter settings (already interpolated if animatable). You can store a handle to any additional non-parameter data in `VideoHandle.InstanceData`. If you do so, deallocate the handle in response to *fsDisposeData*, or your plug-in will leak memory.

The video your filter receives may be interlaced, in the field order determined by the project settings. If interlaced, your plug-in will be called twice for each frame of video, and each `PPix` will be half the frame height.

fsDisposeData

Optional. Called when the project closes. Dispose of any instance data created during *fsExecute*. See `VideoHandle->InstanceData`.

fsCanHandlePAR

Optional. Indicate how your filter wants to handle pixel aspect ratio by returning a combination of the following flags.

This selector is only sent if several conditions are met. The pixel aspect ratio of the clip to which the filter is applied must be known, and not be square (1.0). The clip must not be a solid color. The PiPL bits `anyPixelAspectRatio` and `unityPixelAspectRatio` must not be set.

Flag	Description
`prEffectCanHandlePAR`	Premiere should not make any adjustment to the source image to compensate for PAR
`prEffectUnityPARSetup`	Premiere should render the source image to square pixels during *fsSetup*
`prEffectUnityPARExecute`	Premiere should render the source image to square pixels during *fsExecute*

fsGetPixelFormatsSupported

Optional. Gets pixel formats supported. Called iteratively until all formats have been given. Set

`(*theData)->pixelFormatSupported` to a supported pixel format, and return `fsNoErr`. When all formats have been described, return `fsBadFormatIndex`. See the field-aware video filter sample for an example.

fsCacheOnLoad

Optional. Return `fsDoNotCacheOnLoad` to disable plug-in caching for this filter.

Return Codes

Return Code	Reason
`fsNoErr`	Operation has completed without error.
`fsBadFormatIndex`	Return from *fsGetPixelFormatsSupported* when all pixel formats have been enumerated.
`fsDoNotCacheOnLoad`	Return from *fsCacheOnLoad* to disable plug-in caching for this filter.
`fsHasNoSetupDialog`	Return from *fsHasSetupDialog* to disable setup button in Effect Controls panel
`fsUnsupported`	The selector is not recognized, or unsupported.

VideoRecord

A video filter is passed a handle to a `VideoRecord` with almost every selector.

```
typedef struct {
     PrMemoryHandle           specsHandle;
     PPixHand                 source;
     PPixHand                 destination;
     csSDK_int32              part;
     csSDK_int32              total;
     char                     previewing;
     void *                   privateData;
     VFilterCallBackProcPtr   callBack;
     BottleRec *              bottleNecks;
     short                    version;
     short                    sizeFlags;
     csSDK_int32              flags;
     TDB_TimeRecord *         tdb;
     PrMemoryHandle           instanceData;
     piSuitesPtr              piSuites;
     PrTimelineID             timelineData;
```

```
    char                        altName[MAX_FXALIAS];
    PrPixelFormat               pixelFormatSupported;
    csSDK_int32                 pixelFormatIndex;
    csSDK_uint32                instanceID;
    TDB_TimeRecord              tdbTimelineLocation;
    csSDK_int32                 sessionPluginID;
} VideoRecord, **VideoHandle;
```

specsHandle	Instance settings, persistent across Premiere sessions. Create this handle during *fsInitSpec* or *fsSetup*. Populated by Premiere if the filter's parameters can be manipulated in the Effect Controls Panel. Use Premiere's memory allocation callbacks to allocate memory for the specsHandle.
source	PPixHand for the source video frame.
destination	PPixHand for the destination video frame, always the same size as source. Store the output frame here during *fsExecute*.
part	How far into the effect you are. part varies from 0 to total, inclusive.
total	Total length of the video filter. Divide part by total to calculate the percentage of the time-variant filter for a given *fsExecute*. This value doesn't necessarily correspond to frames or fields.
previewing	Unsupported
privateData	Data private to Premiere. Pass to the frame-retrieval callback when requesting a frame.
callBack	Pointer to VFilterCallbackProcPtr, used for retrieving frames (or fields, for interlaced video) from source clips.
bottleNecks	Pointer to Premiere's bottleRec functions.
version	Version of this structure (kVideoFilterVersion). Premiere Pro CS5 = VIDEO_FILTER_VERSION_11 Premiere Pro CS3 = VIDEO_FILTER_VERSION_10
sizeFlags	Field-rendering information.
flags	If doing a lower-quality render, Premiere will pass in kEffectFlags_DraftQuality during *fsExecute*. The filter can then optionally render a faster, lower-quality image for previewing.
tdb	Pointer to a time database record describing the sequence timebase.
instanceData	Handle to private instance data that persists across invocations. Allocate the memory for this during *fsExecute* and deallocate during *fsDisposeData*. Do not use this field during fsSetup.
piSuites	Pointer to callback piSuites.

`timelineData`	Only available during *fsInitSpec* and *fsSetup*. This opaque handle to the timeline database is required by `timelineFuncs` callbacks available in `piSuites`. This handle is useful in order to have a preview in a modal setup dialog during *fsSetup*.
`altName`	Unused.
`pixelFormatSupported`	Only valid during *fsGetPixelFormatsSupported*. Return pixel type supported.
`pixelFormatIndex`	Only valid during *fsGetPixelFormatsSupported*. Index of fourCC of pixel type supported.
`instanceID`	The runtime instance ID uniquely identifies filters during a session. This is the same ID that is passed to players in `prtFilterRec`.
`tdbTimelineLocation`	A time database record describing the location of the filter in sequence. Only valid during *fsInitSpec* and *fsSetup*.
`sessionPluginID`	This ID should be used in the File Registration Suite for registering external files (such as textures, logos, etc) that are used by a plug-in instance but do not appear as footage in the Project Panel. Registered files will be taken into account when trimming or copying a project using the Project Manager.

VFilterCallBackProcPtr

Pointer to a callback for retrieving frames (or fields, for interlaced video) from the source clip. Do not expect real-time performance from this callback.

```
typedef short (CALLBACK *VFilterCallBackProcPtr)(
     csSDK_int32     frame;
     PPixHand        thePort;
     RECT *          theBox;
     Handle          privateData);
```

Parameter	Description
`frame`	Frame requested. The frame value passed in should be frame * `samplesize`. The callback will always return the current field (upper or lower) during field rendering.
`thePort`	`PPixHand` where Premiere will store the frame
`theBox`	Bounds of the frame you want Premiere to retrieve.
`privateData`	`Handle` provided by Premiere in `VideoRecord.privateData`

sizeFlags

For `sizeFlags`, the following bit flags are of interest:

Flag	Description
gvFieldsEven	The video filter should render upper-field dominance
gvFieldsOdd	The video filter should render lower-field dominance
gvFieldsFirst	The video filter is currently rendering the dominant field

Additional Details

Fields and Field Processing

In an interlaced project, Premiere calls your video filter once per field. This allows video filters to have interlaced motion. `(*theData)->total` will be twice as large, each frame will be half-height, and rowbytes will double.

Respect the value of `rowbytes` when traversing data or the output will be incorrect.

Frame Caching

The rendered output of video filters is stored in the host media cache. For example, when the user scrubs over a frame with a filter on it, the filter will be called to render its effect on the frame and return the buffer to Premiere. Premiere caches the returned frame, so when the user scrubs over the same frame, Premiere will return the cached frame without having to call the filter again. If the user has modified the filter settings, the clip settings, the preview quality, etc, Premiere will call the filter to render with the new settings, but will keep the previously cache frame for a while. So if the changes are reversed, Premiere may still have the cached frame to return when appropriate.

If the filter should generate random, non-deterministic output, or if it changes over time without keyframes, the randomness bit must be set in the `ANIM_FilterInfo` section in the PiPL (.r file). If you set the bit to `noRandomness`, Premiere will only render one frame of a still image.

Creating Effect Presets

Effect presets appear in the Presets bin in the Effects panel, and can be applied just like Effects with specific parameter settings and keyframes. Effect presets can be created as follows:

1) Apply a filter to a clip
2) Set the parameters of the filter, adding keyframes if desired
3) Right-click on the filter name in the Effect Controls panel, and select "Save Preset..."
4) Create preset bins if desired by right-clicking in the Effects panel and choosing "New Presets Bin"
5) Organize the presets in the preset folders
6) Select the bins and/or presets you wish to export, right-click, and choose "Export Preset"

On Windows, newly created presets are saved in the hidden Application Data folder of the user's Documents and Settings (e.g. C:\Documents and Settings\[user]\Application Data\Adobe\Premiere Pro\[version]\Effect Presets and Custom Items.prfpset). On Mac OS, they are in the user folder, at ~/Library/Application Support/Adobe/Premiere Pro/[version]/Effect Presets and Custom Items.prfpset.

Effect Presets should be installed as described in the section, "Plug-in Installation". Once they are installed in that folder, they will be read-only, and the user will not be able to move them to a different folder or change their names. User-created presets will be modifiable.

Effect Badging

Starting in CS5, video filters now appear with badges in the Effects panel to advertise if they support YUV, 32-bit, and/or accelerated rendering. The user can filter the list of effects to show only the effects that support those rendering modes. Video filters will automatically receive YUV and 32-bit badges if they advertise support using the existing *fsGetPixelFormatsSupported*. Custom badges can also be created.

To add your own effect badge, go to the following folder:
On Windows: `[App installation path]\Settings\BadgeIcons\`
On Mac OS: `Adobe Premiere Pro CS5.app/Contents/Settings/BadgeIcons/`
In that folder are the PNG graphics that are loaded at runtime for various badges, and an additional '`Sample.png`' and '`Sample.xml`' file.

1) Copy the `Sample.png` file to a new name that matches whatever you want to call the new badge (like '`NewBadge.png`'). Edit the PNG as you'd like, but don't change the image dimensions.
2) Copy the `Sample.xml` file to a new name that matches whatever you want to call the new badge (like '`NewBadge.xml`'). Edit the list of match names that you want to be decorated with your badge. Change the `<Name>` tag to the name you chose in step 1 (like '`NewBadge`'). You can also add your tooltip text as the `<DescriptionItem>` tags. These tags act as a localization map with the langid as the key. If a language isn't found, '`en_US`' is used by default.
3) Relaunch the application. You'll get a badge filter icon next to the others and a badge icons next to each effect that was listed in the XML file.

Note: 'Sample' is a special case that is intentionally excluded. Any other *.xml/*.png pair will be used.

Real-Time Video Filters

In order to allow for real-time previews in video filters, an opaque handle to the current timelineData is passed to the video filter during *fsSetup* for use with piSuites->timelineFuncs.

Use this handle to obtain a real-time preview, provided by a player (provided you can communicate directly with the player; not every video filter can be made real-time).

Premiere Elements and Effect Thumbnail Previews

Premiere Elements (but not Premiere Pro) displays visual icons for each effect. You will need to provide icons for your effects, or else an empty black icon will be shown for your effects, or even worse behavior in Premiere Elements 8. The icons are 60x45 PNG files, and are placed here:

[Program Files]\Adobe\Adobe Premiere Elements [version]\Plug-ins\Common\EffectPreviews\

The filename should be the match name of the effect, which you specify in the PiPL, prefixed with "PR." So if the match name was "MatchName", then the filename should be "PR.MatchName.png"

11 Device Controllers

Device controllers control hardware devices such as cameras and tape decks using various communication protocols. They are called by Premiere (to interact with video hardware) from the Capture panel and Export to Tape. They set hardware operating modes, tell Premiere what mode the hardware is in, and work along with recorders to provide Premiere with timecode information from the hardware. Export to Tape is performed using insert edits, where the device controller directs Premiere to insert a video segment at a specified timecode. With Export to Tape, Premiere drives the edit, using the device controller perform the operation.

If you've never developed a device controller before, you can skip the What's New sections, and go directly to Getting Started.

What's New in Premiere Pro CS3?

The timebase is now set to device controllers as `preferredScale` and `preferredSampleSize` in `DeviceRec`. This is more specific than the old `timerate` value. Use this rather than calling `piSuites->utilFuncs->getSettings(kSettingsProjectScale)` or `getSettings(kSettingsProjectSampleSize)`.

Getting Started

You'll need a thorough understanding of the device(s) you hope to control before developing a Premiere plug-in. Begin with the sample project, progressively replacing its functions with your own.

Resources

Device controllers use a basic PiPL to specify their name and the match name that Premiere uses to identify them. When making changes to the PiPL resource, rebuild the plug-in each time, so that the PiPL will be recompiled.

Entry Point

```
short xDevice (
     short          selector,
     DeviceHand     theData)
```

selector is the action Premiere wants the device controller to perform. `DeviceHand` provides all pertinent information. Return `dmNoError` if successful, or an appropriate return code.

Selector Table

This table summarizes the various selector commands a device controller can receive.

Selector	Description
dsInit	Create data structures, choose an operating mode.
dsSetup	Display your modal setup dialog.
dsExecute	Perform a specified device control command.
dsCleanup	Dispose of any allocated data structures.
dsRestart	Restart device controller – used at startup to reconnect to a device.
dsQuiet	Disconnect from the device, but don't dispose of allocated structures.
dsHasOptions	New in Premiere Pro 2.0. Return `dmHasNoOptions` to disable the device controller options button.

Selector Descriptions

dsInit

Create a handle for instance data; store it in the `DeviceRec.deviceData`. Choose a default operating mode if more than one is available. A dialog can be presented to prompt the user for settings. Open any necessary drivers, connect to your hardware. See Implementation Tips.

dsSetup

Display your modal setup dialog. If your device controller doesn't require user input, this selector can be safely ignored, but should return `dmNoErr`. There currently is no way to disable the Setup button in the device control options.

dsExecute

Perform a device control operation based on the command in the `DeviceRec`. See the Commands section below for detailed descriptions.

dsCleanup

Disconnect from hardware and dispose of the plug-in's local data (stored in `deviceData`).

dsRestart

Reestablish connections to hardware devices. This selector is similar to *dsInit*, but `deviceData` is already populated.

dsQuiet

Like dsCleanup; disconnect from the device, but don't dispose of local data. dsRestart will be sent to reconnect the device.

dsHasOptions

New in Premiere Pro 2.0. Return `dmHasNoOptions` to disable the device controller options button.

Return Codes

Return Code	Reason
`dmNoErr`	Operation has completed without error.
`dmDeviceNotFound`	The device is not available.
`dmTimecodeNotFound`	The device cannot read the timecode from the media, or there is none to be read.
`dmBadTimecode`	The device has timecode but it doesn't trust it.
`dmCantRecord`	The device is unable to record to the media.
`dmUserAborted`	The operation has stopped because the user cancelled.
`dmLastErrorSet`	The device controller set the last error string using the Error Suite.
`dmExportToTapeFinished`	The device controller is signaling the end of the export to tape operation.

dmTapeWriteProtected	Return value during Export To Tape if tape is write protected.
dmNoTape	Return value during Export To Tape if there is no tape in the deck.
dmLastInfoSet	The device controller set the last info string using the SweetPea Error Suite.
dmLastWarningSet	The device controller set the last warning string using the SweetPea Error Suite.
dmHasNoOptions	Return during *dsHasOptions* to disable the device controller options button..
dmUnknownError	The device controller set the last error string using the Error Suite.
dmUnsupported	The selector is not recognized, or unsupported.
dmGeneralError	Unspecified error.

DeviceRec

A device controller is passed a handle to a `DeviceRec` with every selector.

```
typedef struct {
     PrMemoryHandle          deviceData;
     short                   command;
     short                   mode;
     csSDK_int32             timecode;
     short                   timeformat;
     short                   timerate;
     csSDK_int32             features;
     short                   error;
     short                   preroll;
     CallBackPtr             callback;
     PauseProcPtr            PauseProc;
     ResumeProcPtr           ResumeProc
     long                    xtimecode;
     long                    keycode;
     short                   editmode;
     short                   exteditmode;
     Print2TapeProcPtr       PrintProc;
     HWND                    parentWindow;
     piSuitesPtr             piSuites;
     char*                   displayName;
     TimecodeUpdatePtr       TimecodeUpdateProc;
     void*                   classID;
```

```
    long                        version;
    short                       videoStreamIsDrop;
    short                       autoDetectDropness;
    char*                       currentDeviceIDStr;
    long                        preferredScale;
    unsigned long               preferredSampleSize;
    char                        reserved[36];
} DeviceRec, *DevicePtr, **DeviceHand;
```

`deviceData`	Handle to private data allocated during dsInit; persists across invocations.
`command`	The command being performed when you receive *dsExecute*.
`mode`	Used in three ways. For *dsExecute/cmdNewMode*, mode contains your device's new mode. For *dsExecute/cmdStatus*, mode is where you indicate the device's current mode (the last mode reported will still be there). For *dsExecute/cmdShuttle*, mode contains the shuttle rate (-100 to 100).
`timecode`	Used three ways. For *dsExecute/cmdGoto* and *dsExecute/cmdLocate*, the timecode field indicates the timecode to which your should move. For *dsExecute/cmdStatus*, return the deck's current timecode position via the timecode field, where `kInvalidTimecode` will display "N/A" (not available), -2 will blank the timecode display, and -3 will display "Searching…". For *dsExecute/cmdJogTo*, timecode specifies the location to which you should jog.
`timeformat`	Reports the format of timecode for a *dsExecute/cmdStatus*; 0 for non-drop frame, 1 for drop-frame.
`timerate`	Reports the frames-per-second rate of timecode for a *dsExecute/cmdStatus* call. Set to 24, 25, 30, or 60.
`features`	Reports the device's features in response to a *dsExecute/cmdGetFeatures* call.
`error`	Set this field to an appropriate error code and return a non-zero value from your device controller.
`preroll`	Used by *dsExecute/cmdLocate*. Preroll is how far before (smaller timecode) the seek time specified in timecode you should seek. The preroll value is the product of a calibration sequence the user performs.
`callback`	Pointer to a routine to call during *dsExecute/cmdLocate* to determine if the user is attempting to abort. `typedef csSDK_int32 (*CallBackPtr) (void);` If the return value is non-zero, the user has attempted to abort.

PauseProc	Pointer to a routine that you can call to temporarily pause any sequence grabber operations in a device-controlled window. It is defined as follows: `typedef void (*PauseProcPtr) (void);`

ResumeProc	A pointer to a routine to call to resume sequence capture after calling PauseProc. Every call to PauseProc must be matched by a call to ResumeProc.
	
	`typedef void (*ResumeProcPtr) (void);`
	
	Call these routines before putting up an error alert, for instance:
	
	`(*(*theData)->PauseProc)();`
	`// your error handler here`
	`(*(*theData)->ResumeProc)();`
	
	If PauseProc isn't called before putting up an alert (or any other dialog), video will be played over it

xtimecode	Duration of the movie being exported (used for the Export to Tape).
keycode	Unused.

editmode	Can be any combination of the following flags to enable user actions:
	
	`insertVideo,`
	`insertAudio1,`
	`insertAudio2,`
	`insertTimeCode,`
	`insertAssemble,`
	`insertPreview`

exteditmode	Unused.

PrintProc	A pointer to a plug-in function Premiere calls to print to tape.
	
	`csSDK_int32 (*Print2TapeProcPtr)(`
	`PrMemoryHandle deviceHand,`
	`long selector);`
	
	deviceHand is passed to the plug-in in DeviceRec. selector can be *setupWaitProc*, *idle*, or *complete*.
	
	See *cmdInsertEdit*.

piSuites	Pointer to universal callback suites.
displayName	A 255 character string to display the name of the device the plug-in is currently controlling.

TimecodeUpdate-Proc	During *cmdLocate*, use this to report timecode.
	
	`void (*TimecodeUpdatePtr)(`
	`  csSDK_int32  outTimecode,`
	`  void*  outClassID);`

classID	Used for TimecodeUpdateProc
version	Device controller API version Premiere Pro CS5 - kDeviceControlAPIVersion10 Premiere Pro CS3 and CS4 - kDeviceControlAPIVersion9
videoStreamIs-Drop	New in Premiere Pro 2.0. If autoDetectDropness was set earlier, and the recorder called FormatChangedFunc to provide the drop-frame attribute of the timecode, Premiere will call *cmdSetDropness* and use this to tell the device controller if the video stream is drop-frame.
autoDetectDropness	New in Premiere Pro 2.0. Set this to true if you want Premiere to notify the device controller whether or not the video stream uses drop-frame timecode. Premiere will get this timecode information from the active recorder. The result will be sent during *cmdSetDropness* in videoStreamIsDrop.
currentDevice-IDStr	For internal use only.
preferredScale	New in Premiere Pro CS3. Use this rather than calling piSuites->utilFuncs->getSettings(kSettingsProjectScale).
preferredSample-Size	New in Premiere Pro CS3. Use this rather than calling piSuites->utilFuncs->getSettings(kSettingsProjectSampleSize).

Parameter	Description
frame	Frame (or field, for interlaced video) requested.
thePort	PPixHand where Premiere will store the frame
theBox	Bounds of the frame you want Premiere to retrieve.
privateData	Handle provided by Premiere in VideoRecord.privateData

Commands

When the plug-in receives *dsExecute*, DeviceRec.command indicates the behavior requested.

Command	Description
cmdGetFeatures	Fill in the features field with the device's features.

cmdStatus	Return the deck mode and current timecode position.
cmdNewMode	Change the deck's mode.
cmdGoto	Go to a particular time.
cmdLocate	Go to a particular time and return when you're there.
cmdShuttle	Shuttle the deck at a specified rate.
cmdEject	Eject media.
cmdInsertEdit	Export To Tape
cmdGetDeviceDisplayName	Provide the device display name for display in the Capture Panel.
cmdSetDropness	Tells the device controller whether the current timecode is drop-frame or non-drop-frame.
cmdGetCurrentDeviceIdentifier	For internal use only.

cmdGetFeatures

Populate DeviceRec.features with the features of your device controller, using the following flags:

Flag	Description
fExportDialog	The device controller has an export dialog and wishes to control the edit.
fCanInsertEdit	Insert edit mode is supported.
fDrvrQuiet	Quiet mode is supported.
fHasJogMode	Jog is supported.
fCanEject	Media ejection is supported.
fStepFwd	Stepping the device forward one frame is supported.
fStepBack	Stepping the device backward one frame is supported.
fRecord	Your device can record.
fPositionInfo	Your device can retrieve position information.
fGoto	Your device can seek to a particular frame. You must also set fPositionInfo, and respond to *cmdGoto*.
f1_5	Your device can play at one-fifth speed.
fBasic	Your device supports the basic five deck control operations: stop, play, pause, fast-forward, and rewind.
fReversePlay	Your device can play in reverse.

`fCanLocate`	Your device can accurately locate a particular timecode and supports *cmdLocate*. Please do so; cmdLocate is more accurate than *cmdGoto*.
`fCanShuttle`	Your device is capable of variable-speed shuttle operations, forward and backward.
`fNoTransport`	Device supports no transport modes (play, stop, etc).

cmdStatus

Premiere sends *cmdStatus* to obtain the deck's current mode (play, pause, etc.) and the current timecode position. Store the device's current mode in mode, and the current timecode value in timecode. Be sure to set `timerate` and `timeformat` as described in `DeviceRec`.

The values of mode and timecode persist. If you only know one of the two pieces of information, store it, and ignore the other. If your device controller makes two calls to determine these values, alternately request one and return the other.

cmdNewMode

Puts the device into a new operating mode, specified in `mode`.

Mode	Description
`modeStop`	Stop.
`modePlay`	Play.
`modePlay1_5`	Play at 1/5 speed.
`modePlay1_10`	Play at 1/10 speed.
`modePause`	Pause.
`modeFastFwd`	Fast forward.
`modeRewind`	Rewind.
`modeRecord`	Record.
`modeGoto`	Go to time specified in `DeviceRec.timecode`.
`modeStepFwd`	Step one frame forward.
`modeStepBack`	Step one frame backward.
`modePlayRev`	Play backward at full speed.
`modePlayRev1_5`	Play backward at 1/5 speed.
`modePlayRev1_10`	Play backward at 1/10 speed.
`modeTapeOut`	No tape is in device.
`modeLocal`	Device is unavailable.
`modeRecordPause`	Pause in record mode.

`modeRecordPlayFastFwd`	Fast forward in play mode.
`modeRecordPlayRewind`	Rewind in play mode.

cmdGoto

Seek to the timecode specified by timecode, and place the device in pause mode (if you were able to complete the seek) or stop mode (if there was an error). Often you will set up an asynchronous seek and return immediately.

Premiere send *cmdStatus* requests until the mode is *cmdPause* or *cmdStop*. While seeking, set `mode` to `modeGoto`; Premiere will put "Searching…" in the timecode display of the supervising window. when done seeking, store the new mode (`modePause` or `modeStop`) in `mode`.

cmdLocate

Seek to an exact frame and return immediately with the device in `modePlay`. This is a synchronous operation; do not return until the operation is complete or an error occurs.

`preroll` indicates how far before the specified time in timecode to which to seek. Preroll value is set by the user through calibration.

cmdShuttle

Sent when the user moves the shuttle control; mode is the shuttle speed:

Use intermediate speeds if the device supports them. If it doesn't implement shuttling but does support multiple play speeds, Premiere will simulate shuttling by playing at different rates, based on the shuttle control position. Better results can be obtained by directly supporting shuttling with the *cmdShuttle* command.

cmdInsertEdit

Sent if the device controller supports insert mode and wants to control the edit (set `fExport-Dialog` and `fCanInsertEdit` during *cmdGetFeatures* to do so).

When the user invokes Export To Tape, Premiere prepares to play the chosen clip and sets the following in the `DeviceHand`:

`command` = *cmdInsertEdit*
`mode` = `modeRecord`
`xTimecode` = duration of the movie

Premiere then enters a loop, calling the device controller with the above `DeviceHand`. When the device controller returns, Premiere sends the `PrintProc` specified in `DeviceHand.setupWaitProc`. Premiere will have already performed the preroll; everything is ready to play.

When the device controller returns, Premiere plays the clip, sending idle to `PrintProc` once per frame. Premiere again calls the plug-in's entry point with the `DeviceHand`, allowing the device controller to perform any cue operations. Premiere calls `PrintProc` with complete when finished. If *cmdInsertEdit* is proceeding correctly `PrintProc` should always return 0.

cmdGetDeviceDisplayName

Sent so the device controller can provide the device display name for display in the Capture Panel. The device controller fills in `DeviceRec.displayName`.

cmdSetDropness

Sent only if `DeviceRec.autoDetectDropness` is set to true. This selector tells the device controller whether the current timecode is drop-frame or non-drop-frame, as determined by the active recorder. The timecode information is passed in `videoStreamIsDrop` in `DeviceRec`. Sent when recorder determines drop-frame attribute and calls `FormatChangedFunc`.

Additional Details

Handling dsInit and dsRestart

dsInit must allocate a new `DeviceData` handle; *dsRestart* uses the handle provided. *dsInit* can fall into the *dsRestart* case.