Sony Vegas Video Plug-in SDK

January 2013

Thank you for downloading the Sony Vegas Video Plug-in SDK. This new version of the SDK is based on the OFX standard (http://openfx.sourceforge.net/) for video plug-ins. We recommend that you read through some of the OFX documentation before creating a new video effect plug-in.

If you have further questions, comments, or bug reports regarding the Vegas Video Plug-In SDK you may contact us at: SCS-VideoPIDK@am.sony.com.

Key features of the new interface

1. Temporal frame access.
2. UI is parameter based so you don't need to do UI programming.
3. Allows the user to individually keyframe parameters.
4. Ability to do "analysis" passes from the plug-in UI.

Basics

Parameter-Based UI

One of the key features of the OFX type design is that each effect defines a set of parameters and the host application (in this case Sony Vegas) generates the UI for your plug-in. This should allow you to concentrate on the function of your plug-in instead of designing UI. Basic parameter types that you can define are: doubles, 2D doubles, 3D doubles, integers, 2D integers, 3D integers, Booleans, choices, strings, colors with alpha, and colors without alpha. Each parameter has a name that you use to reference it, a label that is used to show the user in the UI, and optionally a scriptname for localization and a hint that is shown to the user as a ToolTip.

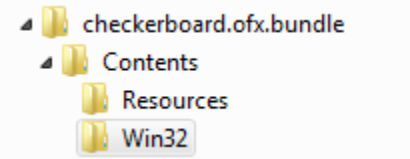
Plug-In ID strings

Previous versions of Vegas plug-ins used a GUID to uniquely identify each plug-in. The new OFX system uses a string based on your reverse domain name and the plug-in name. (i.e. "com.sony.am.checkerboard" or "com.sony.am:colorcorrector").

Plug-In Bundles

OFX plug-ins are bundled as a folder (named something like "youreffect.ofx.bundle") with a subfolder "Contents" and another subfolder for resources (including the localization xml file) named "Resources" and one folder for each platform (i.e. "Win32" or "Win64"). The DLL for your plug-in will have the extension .ofx.

A typical OFX plug-in bundle:



When installing a plug-in, you will need to copy the entire bundle folder to the install location (typically "C:\Program Files\Common Files\OFX\Plugins\").

OFX Feature Support Overview

OFX callback suites

Property Suite	Full Support	
Image Effect Suite	Full Support	
Progress Suite	Full Support	
Timeline Suite	Full Support	
Parameter Suite	Full Support	
Memory Suite	Full Support	
Message Suite	Full Support	V1 and V2
Multi Threading Suite	Full Support	
OpenGL Render Suite	Full Support	
Interact Suite	Not Supported	See Custom UI section for alternative

OFX Context Support

Generator	Full Support	
Filter	Full Support	
Transition	Full Support	
Paint	Not Supported	
Retimer	Not Supported	
General	Partial	For compositing (Vegas can only use compositors with 2 inputs; see note below about defining compositors)

OFX Feature Support

Temporal Frame Access	Supported	
Custom OpenGL Interface	Not Supported	See Custom UI section for alternative

Defining Compositors

To define a compositor effect for Vegas, use the OFX General context and define two input clips and one output clip. The output clip must be named “Output.” The input clips can either be named “Foreground” and “Background” or named something that starts with “Source” (i.e. “Source1” and “Source2” or “SourceA” and “SourceB”).

Sony Vegas Extensions

BGR pixel order

Vegas does its internal processing with pixels in the BGR order instead of RGB, which is standard for OFX. If you can process image frames in BGR order, then a small speedup can be gained. First you need to tell Vegas you can process in BGR order by adding code something like this in the PluginFactory::describe function:

```
desc.addSupportedBitDepth(eBitDepthUByteBGRA);
desc.addSupportedBitDepth(eBitDepthUShortBGRA);
desc.addSupportedBitDepth(eBitDepthFloatBGRA);
```

Then in your Plugin::render function first get the pixel order:

```
OFX::PixelOrderEnum  dstOrder = dstClip_->getPixelOrder();
```

. . . and then process things appropriately. The checkerboard generator example is an example of this.

Custom UI

Vegas allows you to define a window for custom UI if you need special controls or input that the basic parameter types do not allow for. To do this, you need to define an Interact class that inherits from the OFX::HWNDInteract class. In this class you will need to override the four functions: createWindow, moveWindow, disposeWindow, and showWindow. Then in the PluginFactory::describe function set a new instance of your Interact class in the image effect description:

```
desc.setHWNDInteractDescriptor(new YourInteractDescriptor);
```

An example of all this is in the kitchen sink generator example.

Help File

Vegas has a “Help” button for the plug-in by the UI parameters. By default, this will show a help file (either .html or .chm that matches your plug-in’s name in your plug-in bundle’s resource directory). You can override this and display your own help by overriding the “invokeHelp” function in your plug-in class. You can also change what the name of the file is by setting it in your plugin::describe function with desc.setHelpPath(“path\relative\to\resource\or\absolute.html”).

About

Vegas has an “About” button for the plug-in by the UI parameters. By default, this about dialog will show the plug-in name, grouping, version, and description settings. The description setting can be set in

your plugin::describeInContext setting with desc.setPluginDescription("Your plug-in description here."). You can override this and display your own about dialog by overriding the "invokeAbout" function in your plug-in class.

Presets

Vegas allows for plug-ins to keep a set of preset files for each plug-in. Vegas searches two locations for preset files for a given plug-in. The first location is in the plug-in bundle directory of the form: "plugin.bundle.ofx\Contents\Presets\com.yourcompany.pluginID\pluginContext" where the pluginContext is one of: "Generator", "General", "Filter", "Transition", "Retimer", or "Paint". The second directory Vegas looks for presets is "User\Documents\OFX Presets\com.yourcompany.pluginID\pluginContext". The presets in the bundle directory Vegas will consider read-only and won't allow the user to change or delete them. The user documents directory is where Vegas will put new user-created presets and will allow those to be edited and deleted.

The easiest way to create presets is to run Vegas, bring up your plug-in, edit the values, and save the preset (type the name in the preset combo box and click the Save button). When you have come up with a set of presets that you like, you can then copy them to your bundle directory and ship them with your plug-in.

Localization

In addition to checking the common resource file (plugin.bundle.ofx\Contents\Resources\plugin.xml), Vegas will also check for split localized files for each locale of the form plugin.bundle.ofx\Contents\Resources\plugin.ll-CC.xml where ll-CC is the locale identifier. (Sometimes separate files are easier for localization vendors to process.) Vegas is currently localized for the following: English, de-DE (German), es-ES (Spanish), fr-FR (French), and ja-JP (Japanese). Also note in the 'ofxLocale' attribute of the 'OfxResourceSet' item in the resource file Vegas will look to match locales based on ll-CC format.

Both the checkerboard generator and the kitchen sink are examples of localization.

Stereoscopic Rendering

Two arguments have been added to the render action arguments structure to contain stereoscopic rendering information. They are "viewsToRender" and "renderView". During typical processing "viewsToRender" is set to 1 and "renderView" is set to 0. During stereoscopic rendering, "viewsToRender" will be set to 2, rendering will be called twice, once with "renderView" set to 0 (left eye) and once with "renderView" set to 1 (right eye).

Vegas Search Path for OFX Plug-Ins

Vegas supports all the standard search locations for OFX Plug-ins as defined in the 1.0 specification:

1. Semicolon-separated list of paths in the OFX_PLUGIN_PATH environment variable
2. (program files common)\OFX\Plugins (ie "c:\Program Files\Common Files\OFX\Plugins")
3. C:\Program Files\Common Files\OFX\Plugins

In addition, Vegas will also search the “OFX Video Plug-Ins” subfolder below the Vegas application folder, but please note that this location is reserved only for plug-ins that ship with Vegas.

Using Plug-Ins in Vegas

Vegas supports several different contexts to place filters as well as support for generators, transitions, and compositors.

Media Generators

1. Run Vegas.
2. Open the Media Generators window (View->Media Generators). If your media generator was found by Vegas, it will show up in this list.
3. Select your generator (if your render function is called and working a thumbnail called “(Default)” will be generated with the default parameters for your generator.
4. Drag your generator name or the thumbnail to the timeline.
5. The Video Media Generator window will appear with your generator’s parameters and allow you to modify the values.

Media Filters

1. Run Vegas.
2. Add media to the timeline (either with a generator or by dragging a video clip to the timeline)
3. Right-click the event and choose Media FX…
4. The Plug-In Chooser window will appear to select which filter plug-ins to add. Add your filter.
5. Click OK.
6. The Media FX window will appear with your filter’s parameters and allow you to modify the values.

Event Filters

1. Run Vegas.
2. Add media to the timeline (either with a generator or by dragging a video clip to the timeline).
3. Click the Event FX button on the event (or right-click the event and choose Video Event FX….
4. The Plug-In Chooser window will appear to select which filter plug-ins to add. Add your filter.
5. Click OK.
6. The Video Event FX window will appear with your filter’s parameters and allow you to modify the values.
7. Note that Event filters can be pre-crop (at media resolution) or post-crop (default, at project resolution) and this order can be changed by re-ordering the plug-in chain by dragging elements around.

Track Filters

1. Run Vegas.

2. Add media to the timeline (either with a generator or by dragging a video clip to the timeline).
3. On the left side of the video timeline is the track header (track list) for the video track. Click the Track FX button  .
4. The Plug-In Chooser window will appear to select which filter plug-ins to add. Add your filter.
5. Click OK.
6. The Video Track FX window will appear with your filter's parameters and allow you to modify the values.
7. Note that Track filters can be pre-composite (default) or post-composite and this order can be changed by re-ordering the plug-in chain by dragging elements around.

Project Filters

1. Run Vegas.
2. Add media to the timeline (either with a generator or by dragging a video clip to the timeline).
3. In the Video Preview, click the Video Output FX Button  (or choose Tools->Video->Video Output FX)
4. The Plug-In Chooser will appear to select which filter plug-ins to add. Add your filter.
5. Click OK.
6. The Video Output FX window will appear with your filter's parameters and allow you to modify the values.

Transitions

1. Run Vegas.
2. Add two events to a single track on the timeline (either with a generator or by dragging video clips to the timeline).
3. Drag the events so that they overlap.
4. Right-click the transition area created by the overlap and choose Transition->Insert Other….
5. The Plug-In Chooser window will appear to select which transition plug-ins to add. Add your transition.
6. Click OK.
7. The Video Event FX window will appear with your transition's parameters and allow you to modify the values.

Compositors

1. Run Vegas.
2. Add two events to two different video tracks (either with a generator or by dragging video clips to the timeline).
3. On the left side of the video timeline is the track header (track list) for the video track. Click the Compositing Mode button  .
4. A menu will appear. Choose Custom.
5. The Plug-In Chooser window will appear to select which compositor plug-ins to add. Add your filter.

6. Click OK.
7. The Track Composite Mode window will appear with your compositor's parameters and allow you to modify the values.

Creating a Plug-In from the Examples

Overview

This is a basic quick start to get you into making your own video plug-in. It is a good idea to review the OFX documentation on-line to understand basic principles of the system. After you are more familiar with the system, you can modify it as much as you want.

Steps:

1. Copy one of the existing examples.
2. Change the Plug-in ID.
3. Change the names.
4. Build and test.
5. Define parameters.
6. Connect the parameters.
7. Change the work procedure.

Copy one of the existing projects

Copy one of the existing example projects that fits what type of plug-in (Generator, Filter, Transition, Compositor) you want to make. Update the file names to your new plug-in name.

These example files use a support class library that was made by The Foundry. The class library handles a lot of the underlying details of interfacing with OFX, does error checking, thread processing, and wraps it all up as easy-to-use classes. The source code for this library is in the SonyOfxPIDK\Library folder.

Change the Plug-In ID

Every plug-in needs a unique plug-in ID string. The standard is to use your reverse domain name, a dot or colon, and the plug-in name. (i.e. "com.sony.am.checkerboard" or "com.sony.am:colorcorrector"). In the main .cpp file of the program, go to the end of the file and change the plug-in ID to your unique ID. (The other two numbers to the PluginFactory class constructor are the major and minor version numbers of your plug-in. In the examples these are set to 1 and 0 – version 1.0.)

Change the Names

Change the names of the classes in the .cpp file to names for your plug-in (i.e. change "CheckerboardExamplePluginFactory" to "SparklesPluginFactory" and "CheckerboardPlugin" to "SparklesPlugin", etc.).

Find the “describe” function of the ___PluginFactory class. In there is a call to “desc.setLables” that sets the user-visible name of your plug-in.The function also includes a call to “desc.setPluginGrouping” that sets the visible group your plug-in will be included.

Building and testing your plug-in

Add the project you created to the Visual Studio solution file provided. You will have to add the “ofxsupport” project as a dependency to your project so right click on your project and select “Project Dependencies…” in Visual Studio. In the “Dependencies” tab check “ofxsupport” – this will make sure Visual Studio will build the supporting ofx library first.

After building, find your plug-in bundle (should be the “Debug-Win32” folder where the “ExamplePlugs” solution file is). Copy the plug-in bundle directory (i.e. “checkerboard.ofx.bundle”) to C:\Program Files\Common Files\OFX\Plugins.

Run Vegas and see if your plug-in shows up in the appropriate plug-in list (Media Generators, Video Filters, Transitions…) for that type of plug-in (see Using Plug-Ins in Vegas above).

Vegas will create a log file when it is groveling for plug-ins called $(UserFolder)\AppData\Local\Sony\Vegas Pro\10.0\svfx_video_grovel_x86.log if it is a Win32 build or named svfx_video_grovel_x64.log if it is a Win64 build.

Debugging your plug-in

In your Visual Studio solution file, right-click your project and select “Properties”. Go to “Configuration Properties->Debugging.” For “Command” select “Browse…” and find your vegas100.exe application. Select “Apply”.

Place a breakpoint on the ___PluginFactory class “describe” or “describeInContext” function. Press F5 to run Vegas and wait for the breakpoint to hit.

Turning on detailed logging for your plug-in

You can turn on detailed logging of your plug-in in Vegas by going to Options->Preferences and holding down the Ctrl and Shift keys while you select the menu item. Holding down these keys will make Vegas show a special “Internal” preference page. Select the Internal preference page and search for “OFX Detailed Plugin Log” by setting “Show only prefs containing” to “OFX”. In this field enter in a semi-colon delimited list of the plug-ins you wish to have detailed logging of (i.e. “com.sony.am.checkerboard; com.sony.am.dot”).

The log file will be $(UserFolder)\AppData\Local\Sony\Vegas Pro\10.0\svfx_Ofx1_1_plugin_x86-'***plug-in ID***' (***plug-in index in dll***, ***plug-in context***).log.

Define Parameters

Find the “describeInContext” function of the ___PluginFactory class. Here you define the parameters (doubles, 2D doubles, 3D doubles, ints, 2D ints, 3D ints, flags/Booleans, colors without alpha, colors with alpha, etc.) that will be available for users to adjust for your plug-in. Note each parameter has a **name**

(used to define it and access its value) and a **label** (what is visible to the user) and may also have a **scriptname** (for localization) and a **hint** (shown to the user as a ToolTip).

Connect up getting parameters

After you have defined the parameters, you will then have to connect them up so that your work procedure can get their values.

Find the "__Plugin" class definition in the .cpp file. Add "OFX::___Param" variables for each of the variables you defined before. In the "__Plugin" class constructor, connect the variables to the parameter **names** you gave them before. (i.e. " width_ = fetchDoubleParam("Width");").

Find the "setupAndProcess" function of the "__Plugin" class. This is called to process frames and here you will have to get the values of your variables for that frame. Fill in getting all the values for your variables at the process time and set them in your PluginBase class that is passed into this function (i.e. "processor.setWidth(width_->getValueAtTime(args.time));").

Change the work procedure

Find the "__PluginBase" class that is based off of the OFX::ImageProcessor class. In this class, define the variables to hold your parameters, initialize them in the constructor, and if you want define access/setting functions to isolate them further.

Find the template class (for example, "template <class PIX, int nComponents, int max>") that is based off the "__PluginBase" class. In this class is the "multiThreadProcessImages" function; modify this function to do your image processing.

For Previous Vegas Plug-In Developers

Major changes

1. You define parameters and Vegas generates the UI for your plug-in, instead of designing your own UI for your plug-in. There is a mechanism for adding your own UI panel as well if you need something more specialized.
2. Plug-ins have a unique ID string instead of a GUID.
3. Interfacing to Vegas is through a simple DLL interface and not COM.
4. OFX plug-ins are bundled as a folder (see above)

Steps for converting previous Vegas Plug-ins

1. Create a new project based on one of the example plug-ins (Generator, Filter, Transition, or Compositor). See above for how to do that.
2. Update the name and unique ID to your plug-in and fill in the section on creating parameters. (Review the Roadmap from Scratch document that walks you through all those steps).
3. Copy over your work proc from your previous plug-in.

Providing an uplift path for your previous plug-in

Vegas allows you to specify that your plug-in is an upgrade to a previous Vegas plug-in.

1. In the describeInContext function add a line telling Vegas what previous plug-in you are upgrading by GUID “desc.addVegasUpgradePath(“{your GUID here}”);”
2. Add an upgradeKeyframe function to your “__Plugin” class. In this function, the input args contains a list of keyframes in memory from the previous plug-in. Interpret them as your previous plug-in would have and add keyframes by the standard OFX interface for each of your parameters.