Vegas Scripting FAQs

Author: Paul Manly
Revision Date: Apr. 21, 2004

Section 1: General

Section 2: Tracks and Events

Section 3: Envelopes

Section 1: General

1.1: What is Vegas Scripting?

When we speak of scripting in Vegas 4, we're not referring to what may first come to mind for many of you, a written description of the scenes and dialog in a video production. Rather, we're referring to programatic access to the internal data structures in Vegas projects that allows you to automate things that you might normally do via the GUI. This may help you perform repetitive tasks, integrate with external applications, and implement customized features.

1.2: What script languages does Vegas use?

Vegas uses the .NET framework to provide scriptability. What this boils down to is that you can write scripts in JScript .NET or Visual Basic .NET. To use the scripting feature, you must first install the .NET Framework which is best done through Microsoft's Windows Update service.

Although JScript and VB are relatively easy to learn, there's no denying it certainly helps to have some programming experience when writing scripts.

Microsoft's developers' web site (among others) has Documentation for JScript.NET and Visual Basic .NET

1.3: Do I need to be a programmer to write scripts?

For the most part, yes. Although JScript .NET and VB .NET are relativly high level languages, it will be very difficult to write effective scripts without some computer programming experience. In fact, often scripts can be more difficult to write than typical C or C++ applications because the development tools and debugging facilities are not as advanced. However, many scripts can be just slightly tweaked to meet your needs and non-programmers should not be afraid to play around with them to alter and enhance their behavior.

1.4: How do I create a script for Vegas?

Scripts are just plain text files with a .js or .vb extension so any text editor will do the trick. Notepad may be the best choice among standard Windows applications.

1.5: How do I use a script that has been posted on the web as a .txt file?

You can copy the entire contents of the script file from your browser window and paste it into a new document in your text editor. Using Notepad, you can do a "Save As" and specify the script file name. You can name it anything you want but make sure to name the file with the .js or .vb extension (depending on the programming language used in the script). Also make sure to choose "All Files" as the "Save as type", otherwise Notepad will append a .txt extension to the file name.

An nice alternative to using copy and paste is to right click in the browser window and choose "View Source" which will launch Notepad with the contents of the script. Then just do a "Save As" as directed above.

1.6: Do scripts pose a security risk to my computer?

Most certainly YES. A script run by Vegas has the power to do almost anything: delete files, read files, write files, execute programs, access the Internet, access file on your LAN, etc. You should always examine the contents of a script before running it and, if you don't understand what it is really doing, you should not run it unless it comes from a trusted source. In general, you should take the same precautions you would take for any executable program you download from the Internet or receive in an e-mail attachment.

1.7: How do I use my own .NET assembly from within a script?

In order for scripts to reference .NET classes, the appropriate assemblies must be loaded into the script engine. By default, Vegas loads the following assemblies for every script that runs:

- mscorlib.dll
- System.dll

- System.Drawing.dll
- System.Windows.Forms.dll

These assemblies cover most of the classes that scripts will use. However, occasionally a script will need to use classes that are not defined in the assemblies above, such as System.Xml, System.Data, or any third party assembly including ones you've written yourself. In these cases, scripts must have a configuration file that tells Vegas which extra assemblies to load.

Each script can have a configuration file which specifies extra .NET assemblies that are referenced in the script. The configuration file is an XML file with the same file name as the script except it has the '.config' extension appended. For example, if your script is named "MyScript.js", its configuration file should be named "MyScript.js.config". The configuration file must be in the same directory as the script.

The following provides an example of a configuration file for a script that references classes in the `System.Xml` namespace:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ScriptSettings>
  <AssemblyReference>System.Xml.dll</AssemblyReference>
  <AssemblyReference>C:\MyAssemblies\VegasHelpers.dll</AssemblyReference>
</ScriptSettings>
```

The root XML element of the configuration file is named `ScriptSettings`. This element can contain any number of `AssemblyReference` elements. The inner text of each `AssemblyReference` element contains the file name of an assembly DLL. For assemblies that are in the GAC (Global Assembly Cache... this includes most assemblies that ship with the .NET Framework), a full path is not required. However, for assemblies that are not in the GAC, either the full path must be specified or the `IsLocal` attribute should be set to `true`. The `IsLocal` attribute, when set to true, tells Vegas that the assembly DLL is located in the same directory as the script. This allows you to distribute your script, its configuration file, and your extra .NET assemblies without hard coding any paths... just make sure your users keep them all together in the same directory. The following provides an example of using the `IsLocal` attribute:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ScriptSettings>
  <AssemblyReference IsLocal="true" >MyHelperAssembly.dll</AssemblyReference>
</ScriptSettings>
```

One problem in the current release of Vegas is that VB scripts can not access assemblies in that are not either in the GAC or in the same directory as the Vegas 4.0 executable (vegas40.exe). It is possible, however, to add your assembly to the GAC by using the gacutil program (See the .NET Framework documentation for more details). Alternatively, you can place a copy of your assembly in the Vegas install directory (which contains vegas40.exe) and use an `AssemblyReference` element similar to one that refers to an assembly in the GAC.

1.8: Is there a way to tell what is the path of currently running script?

Yes, there is a global variable (similar to the global `Vegas` variable) named `ScriptFile` which is the full path of the currently executing script. So if your script needs access to a helper file that resides in the same directory, your code might look like this:

```
import System.IO;

var scriptDirectory = Path.GetDirectoryName(ScriptFile);
var helperFile = Path.Combine(scriptDirectory, "HelperFile.ext");
```

1.9: How do I add a script to the Script Menu?

You can add a script to the Script Menu (under Vegas' Tools menu) by placing the script in the "Script Menu" directory which is located in the Vegas install directory. The script will appear in the menu the next time you launch Vegas or after you select the "Rescan Script Menu Folder" menu item.

1.10: How do I have a custom icon appear for a script in the the toolbar?

You can have a custom icon appear for a script you've added to the toolbar by placing a PNG image in the "Script Menu" folder that has the same name as the script with '.png' appended. For example, if the script is named "My Script.js", the toolbar image should be named "My Script.js.png". The image can be any size but it will be scaled to 16 X 16 pixels.

1.11: My question is not answered in the FAQ, where can I get more help?

The Scripting Forum is a good place to ask questions. The forum is frequently monitored by Sony engineers and friendly Vegas users who have considerable skills and experience. Questions posted in the forum often make their way into this FAQ.

Section 2: Tracks and Events

2.1: How do I find the currently selected track/event?

You can find the currently selected track or event by iterating through each track and event and examining the `Selected` property. The following functions provide an example of this:

```
function FindSelectedTrack() : Track {
    var trackEnum = new Enumerator(Vegas.Project.Tracks);
    while (!trackEnum.atEnd()) {
        var track : Track = Track(trackEnum.item());
        if (track.Selected) {
            return track;
        }
        trackEnum.moveNext();
    }
    return null;
}

function FindSelectedEvent() : TrackEvent {
    var trackEnum = new Enumerator(Vegas.Project.Tracks);
    while (!trackEnum.atEnd()) {
        var track : Track = Track(trackEnum.item());
        var eventEnum = new Enumerator(track.Events);
```

```
            while (!eventEnum.atEnd()) {
                var evnt : TrackEvent = TrackEvent(eventEnum.item());
                if (evnt.Selected) {
                    return evnt;
                }
                eventEnum.moveNext();
            }
            trackEnum.moveNext();
        }
        return null;
    }
```

Note that Vegas can have multiple tracks and events selected at the same time but these functions only return the first instance found.

2.2: How do I set the fade type (curve) for an event?

Every `TrackEvent` object has `FadeIn` and `FadeOut` properties which give you objects that control the event's ASR and (in the case of `VideoEvents`) transition effects. The `Fade` object has a `Curve` property which can be set to one of the `CurveType` enum values. The following example gives an event fade in curve the fast type.

```
evnt.FadeIn.Curve = CurveType.Fast;
```

One tricky aspect of fades comes into play when two events overlap on the same track. In this case, only the trailing event's `FadeIn` really matters and, to designate the type of fade curve for the leading event, you actually must set the trailing event's `ReciprocalCurve` property. The following code snippet makes a slow curved fade transition between an event (`trailingEvnt`) and any event on the same track that overlaps its leading edge:

```
trailingEvnt.FadeIn.Curve = CurveType.Slow;
trailingEvnt.FadeIn.ReciprocalCurve = CurveType.Slow;
```

2.3: How do I make a one second dissolve at the beginning and end of a video event?

You can add a dissolve (or any transition) to an event using its `FadeIn` and/or `FadeOut`. First you can set the length of the transition to the desired amount like so:

```
evnt.FadeIn.Length = new Timecode(1000);

or

evnt.FadeOut.Length = new Timecode(1000);
```

By default, video events fade (sometimes called "dissolve" in other editing systems) to and from either the track's background color or the overlapping event. Similarly, audio events fade in volume. Alternativly, for video events, you can use a transition effect rather than the default fade. You do this by creating a new instance of a transition effect and assign it to the fade's `Transition` property. To create a new transition

effect, you must first find the appropriate PlugInNode. The following function will recursively search a set of plug-in nodes for one whose name matches the given regular expression:

```
function FindPlugInNode(rootNode : PlugInNode, nameRegExp : RegExp) : PlugInNode {
    if (null != rootNode.Name.match(nameRegExp)) {
        return rootNode;
    } else {
        var children : Enumerator = new Enumerator(rootNode);
        while (!children.atEnd()) {
            var childNode : PlugInNode = PlugInNode(children.item())
            var childMatch : PlugInNode = FindPlugInNode(childNode, nameRegExp);
            if (null != childMatch) {
                return childMatch;
            }
            children.moveNext();
        }
        return null;
    }
}
```

And here's a function that uses `FindPlugInNode` to create a new transition effect given a plug-in name regular expression:

```
function CreateTransitionEffect(nameRegExp : RegExp) : Effect {
    var plugIn = FindPlugInNode(Vegas.Transitions, nameRegExp);
    if (null == plugIn)
        throw "failed to find plug-in";
    return new Effect(plugIn);
}
```

Now you can set the event fade transitions and assign their preset:

```
var fadeInTx = CreateTransitionEffect(/dissolve/);
evnt.FadeIn.Transition = fadeInTx;
fadeInTx.Preset = "Additive Dissolve";
var fadeOutTx = CreateTransitionEffect(/dissolve/);
evnt.FadeOut.Transition = fadeOutTx;
fadeOutTx.Preset = "Additive Dissolve";
```

2.4: How do I add a specific file on the currently selected track in the current position?

The first step is to find the currently selected track which is described in FAQ 2.1. Then, you can get the `Timecode` position by calling `Vegas.Cursor`. Next you should create a new `Media` object from the desired file. Next you must create a new event of the appropriate type (audio or video) and add it to the track. Finally you must create a new `Take` object and add it to the event. This is illustrated in the following code snippet:

```
// edit the following line to suit your needs.
var filename = "D:\\video\\Tape 1 - Clip 001.avi";

var track = FindSelectedTrack();
```

```
if (null == track)
    throw "no selected track";
var cursorTimecode = Vegas.Cursor;
var media = new Media(filename);
if (!media.IsValid())
    throw "media file does not exist: " + filename;
var stream, newEvent;
if (track.IsAudio()) {
    stream = media.Streams.GetItemByMediaType(MediaType.Audio, 0);
    if (null == stream)
        throw "media contains no audio streams";
    newEvent = new AudioEvent(cursorTimecode, stream.Length);
} else {
    stream = media.Streams.GetItemByMediaType(MediaType.Video, 0);
    if (null == stream)
        throw "media contains no video streams";
    newEvent = new VideoEvent(cursorTimecode, stream.Length);
}
track.Events.Add(newEvent);
var take = new Take(stream);
newEvent.Takes.Add(take);
```

2.5: How do I add a text event and set its first key frame to a saved preset?

This is similar to FAQ 2.4. This difference is that, instead of using a file name, you must construct the `Media` object using a generator plug-in. The following function helps you create a `Media` object for a given generator name and preset.

```
function CreateGeneratedMedia(generatorName, presetName) {
    var generator = Vegas.Generators.GetChildByName(generatorName);
    var media = new Media(generator, presetName);
    if (!media.IsValid())
        throw "failed to create media: " + generatorName + " (" + presetName + ")";
    return media;
}
```

The following code will add a "Hot" text event to the selected track and the current cursor position. Again, the `FindSelectedTrack` function from FAQ 2.1 is used.

```
var track = FindSelectedTrack();
if (null == track)
    throw "no selected track";
if (!track.IsVideo())
    throw "no selected track not video";
var cursorTimecode = Vegas.Cursor;
var media = CreateGeneratedMedia("Sony Text", "Hot");
var stream = media.Streams[0];
var newEvent = new VideoEvent(cursorTimecode, stream.Length);
track.Events.Add(newEvent);
var take = new Take(stream);
newEvent.Takes.Add(take);
```

One major shortcoming, however, is that the specific text displayed by the new event can not be specified by

a script. This is because, at the moment, individual effect parameters are not accessable to scripts. Only presets values can be set for a given key frame. So this teqnique is really useful only when you can pre-define presets with the desired text or you can go back and manually change the text.

2.6: How do I script a video event's pan/crop settings?

A video event pan and crop settings can be manipulated using its `VideoMotion` property. The `VideoMotion` object has a `ScaleToFill` property and a `Keyframes` collection. Each `VideoMotionKeyframe` has a `Position` in time (relative to the start of the event). Key frames also have a `Bounds` polygon representing the four corners of the view-port onto the source media. The best way to manipulate this bounds rectangle is to use the key frame's `MoveBy`, `ScaleBy`, and `RotateBy` methods. The key frame's `Center` defines the point at the center of rotation.

The following code snipet moves a video event into view from a position off screen over a two second period.

```
// this avoids a horrible JScript compiler bug
var zero : int = 0;
// get the first event of the first track
var evnt = Vegas.Project.Tracks[zero].Events[zero];
// get the first key frame
var key1 = evnt.VideoMotion.Keyframes[zero];
// create a new key frame at 2K milliseconds.
var key2 = new VideoMotionKeyframe(new Timecode(2000));
// add the new key frame
evnt.VideoMotion.Keyframes.Add(key2);
// get the width of the project
var videoWidth : int = Vegas.Project.Video.Width;
// translate the first key frame to the right (which makes the event
// appear to move in from the left since keyframe bounds represent the
// output view port)
var offset = new VideoMotionVertex(float(videoWidth), float(zero));
key1.MoveBy(offset);
```

2.7: How do I set the opacity of a video event?

Opacity can be set using the `Gain` property of the video event's `FadeIn` object:

```
Vegas.Project.Tracks[0].Events[0].FadeIn.Gain = 0.5;
```

Alternatively, You might prefer to add a composite level envelope (`EnvelopeType.Composite`) to the video track.

Section 3: Envelopes

3.1: How do I find a particular envelope for a given track or event?

Finding an envelope is a matter of iterating through the appropriate `Envelopes` collection and comparing envelope types or names. All tracks have an `Envelopes` property and so do video events (but not audio events). All bus tracks also have an `Envelopes` property. The following function will return an envelope of a particular type in a collection of envelopes (or return `null` if not found):

```
function FindEnvelopeByType(envelopes : Envelopes, type : EnvelopeType) : Envelope {
    var i : int;
    var count : int = envelopes.Count;
    for (i = 0; i < count; i++) {
        var envelope = envelopes[i];
        if (envelope.Type == type) {
            return envelope;
        }
    }
    return null;
}
```

The function above works for all envelopes whose type is included in the `EnvelopeType` enum. It will not work, however, for automation envelopes from audio effects. Alternativly, the following function will find an envelope by name which works for all types of envelope, including audio automation envelopes:

```
function FindEnvelopeByName(envelopes : Envelopes, name : string) : Envelope {
    var i : int;
    var count : int = envelopes.Count;
    for (i = 0; i < count; i++) {
        var envelope = envelopes[i];
        if (envelope.Name == name) {
            return envelope;
        }
    }
    return null;
}
```

The `Envelopes` collection class also has a `HasEnvelope(EnvelopeType type)` method that allows you to quickly determine whether or not an envelope of a particulat type is contained.

3.2: How do I add an envelope to a track or event?

Envelopes can be added to all tracks and bus tracks can have envelopes but only video events have envelopes, not audio events. Also, certain envelope types belong to a specific context and can not be added to others. For example, you cannot add a volume envelope to a video event.

The first step in adding an envelope to a track or event is to create a new envelope of the desired type:

```
var volumeEnvelope = new Envelope(EnvelopeType.Volume);
```

The `Envelope` constructor can take any of the values in the `EnvelopeType` enum. The next step is to add the new envelope to the appropriate `Envelopes` collection:

```
audioTrack.Envelopes.Add(volumeEnvelope);
```

In this case, `audioTrack` can be any audio track that does not already have a volume envelope.

3.3: How do I add points to an envelope?

Points are added to an envelope in much the same way envelopes are added to a track or event. The first step is to get the envelope to which you want to add the points and then create a new `EnvelopePoint` object:

```
var envelopePoint = new EnvelopePoint(new Timecode(1000), 2, CurveType.Sharp);
```

The code fragment above creates a new envelope point positioned one second after the start of the track or event to which it will eventually be added. In terms of the envelope graph you see in Vegas' GUI, the envelope point's position represents its x-coordinate in timecode units (the `Timecode` constructor takes either a number of milliseconds or a timecode string such as "01:15:32:05"). The second argument in the `EnvelopePoint` constructor is its value, or the the horizontal coordinate in the envelope graph. This value must lie somewhere between the envelope's `Min` and `Max` properties. Envelopes also have a `Neutral` property which defines the default or middle value for envelope points. The third argument to the `EnvelopePoint` constructor is one of the `CurveType` enum values and represents how the curve following the new point should ascend or decend to reach the next point in the envelope.

The new envelope point must then be added to the desired envelope's `Points` collection:

```
envelope.Points.Add(envelopePoint);
```

One rule to keep in mind that no two points can have the same position. An exception is thrown if you attempt to add a point with the same position as another. It is illegal to set a point's position to a negative time or, in the case of event envelopes, to a time beyond the end of the event. It is also illegal to set a point's value below or above the envelope's min/max range.