Stupeflix Factory Documentation

Release 1.0

Stupeflix

December 02, 2014

Contents

Stupeflix provides a simple After Effects to Stupeflix Widget workflow. This documentation will guide you through our engine limitations and the supported After Effects features.

This documentation is our little living thing, it regularly evoles and we feed it as regularly as we can. If you have any questions regarding the AE docs, feel free to drop me an email at seb (at) stupeflix [dot] com.

CHAPTER 1

Introduction to Adobe After Effects

Adobe After Effects is a widespread animation and compositing program, or, to put it another way, it's kind of Photoshop for video. If you don't have After Effects yet, or need more information about this product, you can check its dedicated page on Adobe website and download a 30 day trial version.

1.1 Why After Effects?

Animating and compositing graphical designs using XML is time consuming, excessively complicated, and ultimately not rewarding. Not to mention that motion designers or graphic artists simply don't code to make things move. As we were looking for a design environment for our templates, we decided After Effects will be our weapon of choice. Widespread acceptation in the motion design world, ability to script it and convert projects to XML sealed the deal for us.

On top of that, After Effects has a great community, and a sweet learning curve.

1.2 What versions are supported?

We are internally using Adobe After Effects CC 2014, but we accepts projects from CS6 and above.

1.3 Where to learn After Effects

If you are new to After Effects and want to quickly learn the basics, head over to VideoCopilot Basic Training Series . You'll be able to create cool stuff in a matter of hours.

Now if you're already an After Effects artist and want to know what the technical constraints and requirements are to work with Stupeflix, just click the Next button below :)

CHAPTER 2

Creating a Stupeflix Compatible AE Project

So here comes the tricky part: creating a project that is compatible with our render engine. Creating projects for Stupeflix calls for a slight change of design paradigm. I'll try to explain this on this page, along with general technical stuff. You'll find on separate pages more information on selected (and important) topics.

But first things first, we need to talk about what we call widgets, so…

2.1 Let's talk about Widgets

From now I will make a distinction between an After Effects Project and an After Effects Widget.

At Stupeflix, we build videos like Lego objects. For each video theme (aka "template") in the Stupeflix Studio, we have basic bricks for each kind of data the user can input, and then plug them together to create a unique video every time. We can add bricks next to each other, on top of another, regroup them together in a long sequence, and so on. Like Legos, the possibilities are endless. You can try and experience that with our Scrapbook template. Play with the template, add a map, a picture and a text, change their orders, put several pictures one after another, with or without captions. Preview you video between each action you make, you'll see that we are using a limited amount of basic bricks and build your video on the fly with them.

When we design a template for the studio, we create all the basic variations of scenes and transitions, or bricks to go on with the Lego analogy, inside the same After Effects project, to share the graphical assets and to ensure that all our variations share the same look and feel. Then we export each brick separately using this brilliant After Effects Script (created by yours truly) into their own projects.

Even if you're not designing a Studio Template, but a personalized video ad campaign, there are numerous advantages to slice your project into smaller widgets. But we'll see that later.

Because grown-ups don't like being reminded that they cannot play Legos anymore, we are calling these bricks Widgets. That was the end of the Lego analogy.

2.2 Designing Your Widget

Now let's talk about what you can and cannot do in After Effects, and how it interacts with our render engine and XML video description language.

Widgets can contain 2 sorts of elements:

Design elements (like the cork texture of the Scrapbook template) Placeholder elements (like the precomps that receive the user picture)

The only difference between a design element and a placeholder is tagging. In order to "convert" a composition or a text layer into user customizable elements, you have to add a tag. We'll come back later on that subject.

So, what's the big picture when designing an After Effects Widget to be used by Stupeflix ? Well, the short answer is: you see that pretty "Effects" menu in After Effects? Well forget about it. Slightly longer answer: you see that pretty "Effects" menu ? Well, pretty much forget about it for placeholders (not for design elements).

Joke aside, since projects are rendered by our engine, not After Effects, we had to develop our own feature set. The most import feature is animation, so we decided to add effects on a case by case basis. We already support some of them as you can see on our Supported Effects page. When I say support, I mean, available on our engine for real time rendering. You can use any effects (including 3rd parties effects) on design elements as long as you Pre render or Proxy them.

You can find a detailed list of all the features we do not support yet on our Unsupported Features page.

Now, on the bright side, we support almost all of the other features you can find in After Effects: Trackmattes, Expressions and Bézier curves, parenting, and all layer transformation properties such as Position, Rotation, Anchor Point, Scale, Opacity and so on. As disabled layers are not exported (except for trackmattes), be sure to parent layers to enabled layers only.

For text layers there is extra complication. We support the same transformations and effects as any other layer, but we don't support any character animations. And because After Effects doesn't allow the export of leading, kerning and most of the font options (including the layer size, or bounding box) we don't support anything else than paragraph alignment, Font face (one per layer), Font size (one per layer) and Font color (one per layer). Also, don't change the anchor point position on a text layer. Last but not least, as we have to recalculate the layer size on our side, it is not recommended to use a text layer as a parent. You can still use it as a child without problem.

You can find more informations about text layers in the Placeholder and Tagging page.

2.3 Paradigm shift

So what about this Paradigm shift I talked about earlier? The big difference between a normal After Effects project and a Stupeflix After Effects project is rendering.

When you design an AE project to make a video, you usually make the final render once, so you don't really care about rendering time or rendering optimization (I don't know anyone converting expressions into keyframes to reduce rendering time).

In the case of a Stupeflix AE project, once your Widgets are in our system, they will be rendered every time a user uses them. The most important thing for us is speed. We target a faster than realtime rendering in SD, and realtime for HD videos. If you create heavy, complicated Widgets, we won't be able to deliver that speed, and it will impact the user experience.

We have a full help section dedicated to Widget optimization. You'll find examples on how you can drastically reduce render times.

2.4 Camera and 3D layers

We completely support 3D layers and cameras. You have to make sure that any composition featuring 3D layers has a single camera. If you have no camera, or more than one camera in your composition, our AE SxC Validation Script will warn you, and log an error.

When building your 3D world, make sure that your 3D layers are in the correct depth order, front layer on top of the layer stack, back layer at the bottom of the stack. Having two 3D layers separated by less than 2 pixels in depth can produce Z-Fight. Z-Fight is an inelegent side effect where our 3D engine don't know which pixel of the 2 layers is in front because the rounded calculation results are equal. This usually happens when rotating in z-space the layers.

You can of course use 2D layers along with your 3D layers.

We don't support the depth of field of the camera. The trick involving Adjustment Layers to break the 3D world doesn't work with Stupeflix. Collapsing Transformation to propagate a precomposed 3D world into another 3D world doesn't work either.

2.5 Nesting and Precomposing

Nesting and Precomposing is a great feature we totally support (except for Collapsed Transformations). We also use precomposing for graphical placeholders.

But you should be light on your nesting depth, as nesting is heavy on our render engine. We think that you shouldn't go deeper than 3 levels of nesting:

main comp (lvl 1)

precomp (lvl 2) precomp (lvl 3)

Of course, precompositions used as placeholders for graphical elements won't be considered as precomps in our system, they will be replaced by the user assets, so they don't slow down rendering.

Also you have to make sure any precomp or nested comp is using the same framerate as it's parent comp. We only allow 1 framerate per project.

2.6 Some Limitations

As our render engine don't use the same algorithms as After Effects, you might notice some aliasing around the edges of your layers. Activating anti-aliasing on our side would drastically reduce the render performance. We have found a very subtle, but yet significant, workaround to get anti aliasing for free.

We have aliasing issues on the edges of layers, not on the content of the layer itself. If your layer, may it be a footage or a composition, have transparent edges it will be ok.

For that I recommand you to make your precomposition a bit wider to leave some empty space around the edges, and to prepare your graphical assets accordingly using the same trick.

If you have Photoshop, you can import your PNG files in it and just make them 4-6 pixel wider by simply adjusting it's canvas size in the Image menu.

Having slightly larger assets works around another limitation: blur. Of the few effects we support, Blur is the most usefull one. But we are facing a big limitation: we cannot blur outside the boundaries of a layer. So if you have a 50×50 pixels asset and apply a blur of 100 on it, only the small 50×50 pixel asset will be blured, showing it's clear cut edges, thus giving a bad result. If you want your layer to have a larger blur than it's boundaries, either precomp it (but it can produce slowdowns as you can see in the Nesting and Precomposing section above) in a larger comp or enlarge it's canvas size in Photoshop.

We don't support masks. You will have to use track mattes instead.

Hold keyframes are not supported either, be sure to be carefull about this.

Blending modes are not supported.

For every layer, may they be placeholders or design elements, make sure that their In Point and Starting Point are exactly the same. If they are not, this will confuse our engine and might give an unexpected result.

Click here for a full list of Unsupported Features.

2.7 Sneak Preview of the XML Power

Your After Effects Widget will be interacting with our XML language. XML is mainly used as a cement between Widgets: It allows stacking and grouping Widgets amongst other things.

XML is also used to directly modify the behavior of a Widget:

To fill the placeholders with user's data To define the behavior of user's data inside a graphical placeholder (ken burs, stretch, etc. . .) To hide a layer To change a Solid color / transparency

If you want to know how does it works, we have a whole section about Mixing After Effects with XML.

2.8 Exporting

In order for us to convert your After Effects Widget into a Stupeflix Widget, we need you to send us your project. In order to do that efficiently, we need you to follow these easy steps:

- Check that your project is Optimized, properly tagged & keyframed, and fully compatible using our AE SxC Validation Script.
- Make sure your main comp is named 'main' and is in the root folder of the project (if not, you'll have a warning, and the script will take care of it during the export process)
- Select your main composition and go to File > Reduce Project.
- Now go to File > Collect Files, select a folder and choose to export all files.
- If you use custom Fonts, include them in the directory created by the Collect Files command.
- Zip the directory.
- Send it to us.

2.9 Demo Project

We have made a sample projects for you to see how everything is working out on a simili real-world project:

If you have After Effects CS5 or above, please download this full featured project: Demo CS5

You can preview the result in our XML Editor here (remix a copy to see the XML code).

In this project, you can customize the front logo, the front text (the left text is linked to the front text), the movie poster on the right and the color of the front pane. For that you will find 2 graphical placeholders and 1 text placeholder.

The project is a mix of 2D/3D layers spread in several precompositions masked by trackmattes.

CHAPTER 3

Creating a Transition Widgets

A transition widget is created like any other After Effects project, but the way it's used in our engine requires some extra carefullness and involves some extra work.

Transition Widgets have 2 mendatory graphical placeholders:

- One for the input video (tagged IMAGE_01)
- One for the output video (tagged IMAGE_02)

Transitions are always inserted between two Widgets elements, and when our render engine stumble upon one, this is what he does:

- It renders the Widget before the transition
- It renders the Widget after the transition
- It then fill the transition with the rendered Widgets videos

What does this means , it first means that your placeholder will get a rendered version of the in and out Widgets, so transition can't change anything in them. it also means that in order to be seamless, the In placeholder must start fullscreen, and the Out placeholder must finish fullscreen.

It's not easy to explain the behavior with words (and then, not easy to have a clear mind about it when reading this), but if you try this transition project, you'll see in context what it does. You can download it here.

All the magic is happening in the transition precomps, which are the transition widgets.

CHAPTER 4

Placeholder and Tagging

As After Effects was not designed to have dynamic content generated from an external source, we need to specify in the project where the user content will be. We have 2 types of content, graphical content such as maps, pictures and videos, and text content. Both are tagged the same way, but are handled slightly differently.

4.1 Graphical Placeholders

If you want to add a placeholder for a Google Map, a video or a picture, you just need to create a precomposition with the correct dimensions and aspect ratio. Then, nest it into your main comp and tag it. What is important to understand is that this tagged precomposition, aka Placeholder, will act as a "window" to the user data. Any modification you do on it such as rotation, zoom, track mattes, transparency and so on will be applied to the user data. Also be aware that any layers or data inside the precomposition will be completly replaced by the user asset.

The size of the placeholder composition depends on the use you plan to have. Depending of the kind of data you expect to fill it with, you can optimize it:

- Maps are usually square, with a max size of 640×640
- Portrait Pictures usually have a 10/15 ratio
- Landscape Pictures usually have a 15/10 ratio
- Videos usually have a 16/9 ratio, but can also be 4/3 ratio

For picture size, I usually use the main composition width as the width of the placeholder. Like this you can always fit the screen without scaling it. For example, if your Widget is in Full HD resolution (1920×1080) I will use 1920 as the default width for the picture placeholders. As general advice, make sure your placeholder is neither too small, nor too big. Scaling up assets can result in a visible loss of quality (above 200% scale, results can really look terrible). Keeping assets scaled down results to a loss of performance in render time. So it's up to you to find the perfect balance between quality, performance and placeholder size.

If you are doing a personalized video campaign and you control where the user pics come from, you can even get perfectly sized placeholders. For instance, pictures coming from Facebook or Instagram uses fixed size, so the more you know from where the assets comes from, the better it is.

If in your application or in our Studio, users select a bigger or smaller picture, our engine will automatically make it fit in your placeholder. We'll see how to do this, and how to customize this behavior in our Mixing your Project with XML.

4.2 Text Placeholders

If you want to add a text placeholder, simply add a text layer. And then be extra carefull and make sure you carefull apply everything that is in this section.

When creating the text layer be sure to use the simple text tool and not the box text tool. To create a simple text layer, just clic on the composition viewer with the text tool selected, instead of drawing a box with it. If your texts are inside text boxes, they will be slightly moved around depending on the anchor point position.

Also make sure the font size of your text is big enough to be readable, especially in high definition videos where texts are usually designed to be small. Then become unreadable if the video is not viewed at full resolution. A small sized text at 1080p won't be readable at all at 360p. The smaller the characters, the more the compression will harm them.

If your text is in a 3D composition and comes close to the camera, it will get scalled in our engine whereas it's vectorized in AE to keep constant quality. In order to avoid to much quality loss, we recommand using a font size 2 times bigger than what you need, and then scale down the layer to 50%.

Texts are the most difficult layers to export from After Effects as we don't have access to a lot of parameters. This is a list of what we *can't* do with text layers:

4.2.1 Character Panel

- Multiple Fonts or different font parameters on a single layer
- Kerning
- Leading
- Horizontally scale / Vertically scale
- Baseline shift
- Tsume
- Faux Bold and Faux italic
- All Caps
- Small Caps
- Sub/Superscript

4.2.2 Paragraph Panel

- Justify (all variations)
- Indent (all variations)
- Add space (all variations)

To sum it up, on a text layer we only support one font, one font style (color included), and the paragraph alignment (left, right or centered). All layer transformations such as rotation, opacity, position, scale, track mattes & so on are still available.

A big warning about the anchor point: Don't manualy change the anchor point from it's default alignment location or your text will be shifted.

Also note that we don't do Vertical Align yet, so if you define a 3 line text input, we'll always fill it from top to bottom.

As for any other layer, Layer Styles are not supported either!

After Effects doesn't allow us to get the exact bounding box of your text layer, so we have to estimate it back inside our engine. In order to do that, you'll need to fill your text layer until you max it out with characters. To estimate it, our engine simply takes the text layer, reads it, renders it to an image file so then we get the width and height of the text zone.

We recommend using Capital W. Spacing and kerning might be slightly different between After Effects and our render engine, so using W allows us to reduce this shift (by reducing the number of characters) when calculating the bounding box on our side.

Capital W makes the font size looks bigger than it actually is (unless you plan to only use capitalized letter in your text). Before filling your text with W for export, make sure to test it with random text (or Lorem Ipsum) to avoid any readability issues.

In our Mixing your Project with XML page we will see how to change the text color and, to some extent, the font face.

4.3 Tagging your Placeholders

So, how do we tell our engine that this nested composition or this text layer is to be replaced with the user input? We simply tag it. To tag it, you just have to add a marker on the layer. To do that, simply select the layer and press the * key on the numeric keypad (or go to Layer > Add Marker).

Now that you have your layer marked, we need to edit the marker to give it all the data we need.

We are using 2 important text zones in the marker settings:

- Comment: the text here won't do anything apart from being displayed in the timeline. This is recommended for easy referencing inside your project. Not mandatory.
- Flash Cue Point Name: This is where we define the type of placeholder, and its index. This information is mandatory. If not properly filled, your placeholder won't be recognized.

As you can write whatever you want in the comment section, let's focus on the Flash Cue Point Name and see what information needs to be there. We actually support 2 indexed tags:

- TEXT_XX
- IMAGE_XX

TEXT_XX, where XX is an index ranging from 01 to 99, is used to tag a text placeholder. IMAGE_XX, where XX is also an index ranging from 01 to 99, is used to tag a picture, map or video placeholder.

If you want to display several times the same user input, may it be a text or a graphical asset, you can tag different elements with the same index. For this to work, you just need to take care of renaming the layers sharing the same indexed tag with the same name.

For example, if you have 3 different text layers displaying the same user input, even if they are in different compositions with different properties, they will need to have the same layer name, and the same tag index.

Tagging a project comes in the very last steps of it creation. In order to properly index your placeholder, you must have already sliced your project into widgets (if needed), or know how it will be sliced. For each Widget, tags must start at 01 and then go on without skipping any number.

Example with a Widget accepting 2 user texts, and 2 user pictures:

This is correct:

- TEXT_01
- TEXT_02
- IMAGE_01

- IMAGE_02

TEXT and IMAGE tags are correctly indexed.

This is incorrect:

- TEXT_02
- TEXT_03
- IMAGE_01
- IMAGE_02

The TEXT tags doesn't start at 01.

This is incorrect too:

- TEXT_01
- TEXT_03
- IMAGE_01
- IMAGE_02

The TEXT tag TEXT_02 is missing

This is also incorrect:

- TEXT_01
- TEXT_02
- IMAGE_03
- IMAGE_04

Indexes are tag independent. Here IMAGE tags cannot take the index continuity of the TEXT tags indexes.

Tags are usually the first place to look when a user text or picture is missing in the stupeflixed video. When using our validation script, you can check the number of placeholders detected in the Overview Panel.

4.4 Demo Project

You can download our exemple project to see how to tag your project: Demo project CS5 or above

CHAPTER 5

Expressions and Bézier Curves

5.1 Cloud Renderer

Expressions, Bezier curves and Hold keyframes are not natively supported by our engine, but they can easily be turned into linear keyframes while preserving ease.

For layers with Bezier Curves, just Alt-Clic on the property stopwatch to create an expression and press the numeric keypad Enter key. Now select the property and go to:

- **Animation > Keyframe Assistant > Convert Expression to Keyframes**

For properties with expressions, select them and go to:

- **Animation > Keyframe Assistant > Convert Expression to Keyframes**

For both operations, you can select multiple properties.

Then, with all the newly created keyframes selected, crtl (or cmd) click on any of them to turn them into diamond shaped keyframes (standard keyframe) instead of roaving keyframes (round shaped keyframes).

5.2 iOS Renderer

On our iOS Renderer, Bézier curves are natively supported by our engine, so no need to rasterize (bake) them using the above methodology. You'll still need to bake the expressions.

CHAPTER 6

Supported AE Features & Formats

6.1 Supported Effects

We currently support this list of effects:

- Fast Blur
- Exposure
- Directional Blur
- Tint
- Tritone
- Threshold
- Hue/Saturation
- Ramp

6.2 Support Details

6.2.1 Fast Blur

- Bluriness: supported and keyframable
- Blur Dimensions: supported but not keyframable
- Repeat Edge Pixels: ignored

6.2.2 Directional Blur

- Direction: supported and keyframable
- Blur Length: supported and keyframable

6.2.3 Bilateral Blur

- Radius: supported and keyframable
- Threshold: supported but not keyframable

- Colorize: ignored

6.2.4 Exposure

- Channels: Master only, not keyframable
- Master > Exposure: supported and keyframable
- Master > Offset: unsupported
- Master > Gamma Correction: supported and keyframable

6.2.5 Tint

- Map Black To: supported but not keyframable
- Map White To: supported but not keyframable
- Amount to Tint: supported and keyframable

6.2.6 Tritone

- Highlights: supported but not keyframable
- Midtones: supported but not keyframable
- Shadows: supported but not keyframable
- Blend With Original: supported and keyframable

6.2.7 Threshold

- Level: supported and keyframable

6.2.8 Hue/Saturation

- Channel Control: Master, Reds, Greens & Blues supported
- Channel Range: ignored
- Master Hue: supported
- Master Saturation: supported
- Master Lightness: supported
- Colorize: supported
- Colorize Hue: supported but not keyframable
- Colorize Saturation: supported but not keyframable
- Colorize Lightness: supported but not keyframable

6.2.9 Ramp

- Start / End of Ramp: supported but not keyframable
- Start / End Color: supported but not keyframable
- Ramp Shape: supported but not keyframable
- Ramp Scatter: ignored
- Blend With Original: supported but not keyframable

6.3 Supported File Types

6.3.1 Graphical Assets

- Png files
- Jpeg files

6.3.2 Video Assets

- h264 files in *.mp4, *.flv, *.f4v or *.mov container
- QT Animation files
- on2 VP6 files, including Alpha channel, in *.flv container

6.3.3 Font files

If you use a special font file in your project, and if you have the correct licences, we need to install them separatly on our servers. We support *.otf or *.ttf files as long as they don't have any special character in their filename, or in their font name.

CHAPTER 7

Unsupported Features & Formats

As we continuously improve our rendering engine, we try to integrate more and more features. Sadly, the list below is still unsupported in our current engine:

7.1 Features

- Effects (except those mentioned here)
- Masks
- Time Remapping
- Lights Layers
- Shape Layers
- Layer Styles
- Adjustement Layers
- 3D Material Options
- Photoshop Live Layers
- Photoshop 3D Layers
- Blending Modes
- Collapse Transformation
- Motion Blur
- Assets larger than 4096x4096
- Compositions larger than 4096x4096
- Layer In Point different than Layer Start Time
- Vertical Align for text Layers
- Frame blending
- Audio *

7.1.1 iOS Renderer

If you design a Widget for our iOS Renderer you'll have to take those extra exceptions into consideration

- Assets larger than 2048x2048
- Compositions larger than 2048x2048
- We don't support audio files inside an After Effects Project, but audio files can be played inside the XML that will embbed your AE Widget. More info about this here.

7.2 Formats (Static Images)

We don't support these graphical assets inside AE projects (non exhaustive list):

- Photoshop Files
- Illustrator Files
- EPS Files
- Tiff Files

We recommand using jpegs for non alpha assets, and png otherwise.

7.3 Formats (Video files)

We don't support file formats other than h264 (in mp4 encapsulation) for non alpha videos, and On2VP6 (in flv encapsulation) for video with alpha.

- For h264 we do recommand using the High Profile with 5.1 Level, 2 pass VBR with a 5-8Mb bitrate for SD videos, and a 10-15Mb bitrate for HD videos and up to 20Mb for Ultra HD videos.
- For On2VP6 we do recommand a 10Mbit bitrate in VBR encoding, with target quality set to Optimal.

We are using Adobe Media Encoder in house to compress our assets, we do recommand it.

CHAPTER 8

Preping your Project

8.1 Non Linear Template

If you're doing a non linear Template, you project must be spliced up into different Widgets.

When designing a Template, it is important to keep in mind that it is the user who is storytelling his video, not you. Depending on what the user wants, and what the user wants to say, we will choose programmatically one widget or another. The most important aspect of your template is visual coherence. You don't know in what order, or in what combination your widgets will be used, so it's essential that they all have a consistent look & feel, and most importantly, that any widget can be used before or after any other widget.

8.1.1 Widget Recommanded Structure

After designing several templates, we found out that some amount of widget variation is necessary to avoid the feeling of videos looking the same, or looping on the same effects

For that we recommend to create at least the following set of widget and variations:

- A widget containing 1 picture (2 variations)
- An overlay Widget containing 1 text for caption (2 same variations)
- A widget containing 2 pictures (3 variations)
- An overlay Widget containing 2 texts for captions (3 same variations)
- A widget containing 3 pictures (3 variations)
- An overlay Widget containing 3 texts for captions (3 same variations)
- A widget containing 4 pictures (3 variations)
- An overlay Widget containing 4 texts for captions (3 same variations)
- A widget containing 1 short text (2 variations, for titles)
- A widget containing 1 long text (2 variations, for long texts)
- A widget containing 1 map (2 variations)
- An overlay Widget containing 1 text for captions (2 same variations)
- A custom transition widget (2 variations)

If you'd like to make your Template more unique, you can also add special widgets such as:

- Intro Widgets

- Outro Widgets
- Lower 3rd / Upper 3rd Widgets
- Customizable Widgets (Background, borders, vignettes, color scheme, fonts...)

8.1.2 Customizable Widgets

If you want to, you can create what we call "Customizable Widgets". In these Widgets, the placeholders are not only used for showing the user pictures, they are also used to receive decoration assets such as background, borders, custom graphic elements.

These Widgets are resource-hungry, so using them can have an impact on performance. You can choose to make an entirely customizable template (and specify default assets), but rendering time will be impacted, and depending on the complexity of your Widgets, render time can really be lengthened.

Mixing your project with XML can help you achieve something similar. With XML you can hide any layer of your AE widget, change the solid or text colors. We are using this technique in our demo project. Preview it online here, or get the AE project here.

8.1.3 Sub-Styles

You can also create sub-styles for your template. Sub-Styles are just a graphical variation of your template, to express a different mood or genre. For example, we could replace the cork background of the Scrapbook template with a concrete one, and white borders with spray paint to give a totally different feeling. Adding to that a faster camera animation and some hip-hop soundtrack and you'll get the idea.

Sub-styles are "pre-made" customizable Widgets. The only difference is that we don't allow the user to modify them on the fly like customizable Widgets, but just select one or more available style from a dropdown list, in order to get better and faster performance when rendering the video.

8.1.4 Technical Considerations

We recommand a default 5 seconds per item duration. An item can be a text, a picture or a map.

For example, a Widget with 3 pictures will have a 15 second duration. If in your widget you choose to display one picture at a time, these 15 seconds include the transitions between your pictures. So it's not 5 seconds of full display per item, it can be shorter or longer depending on how you organize your Widget. But in the end, it will have to be around 15 seconds for 3 pictures. If you choose to add small captions on top of your pictures, they will still be shown for 15 seconds as items displayed simultaneously don't add-up their durations.

8.2 Linear Template

When you are designing a video for an advertising campaign, unlike a non linear Template, you are doing the story-telling for the user, but you allow him/her to somehow interact with your story by customizing the video.

Usually, such projects are designed as a long video with special parts where user assets can be seen. One could even imagine different paths or scenes for the video depending on some user choices.

In such cases, there is not a predefined Widget structure, it really depends on how your video is conceived. From past projects and experience, it is safe to not slice your video into multiple Widgets. Just keep on project for the whole video, or for each path of the video.

8.2.1 Technical Considerations

On such custom projects, you are free to choose your most suitable framerate. But be sure to have all assets in your project, including compositions, set at the same framerate. It's a common source of error to have footage from different sources at different framerates not conformed with the destination framerate.

8.3 Technical Recommandations

Our render engine is much more efficient with standard 16/9 video ratio, such as 640×360, 960×540, 1280×720 or 1920×1080. Our online XML editor, used for previewing, can only preview correctly 16/9 projects. We of course support other resolutions and other aspect ratios, but we recommend you to work in 16/9 aspect ratio.

Working in native resolution is important. If you provide us a Full HD (1080p) project but only allow your users to create 360p videos, the render won't be as efficient as a lot of processing power will be lost dealing with high resolution assets to be scaled down. We highly recommend you to design and provide us a project that is matching your destination resolution.

For the framerate, we strongly recommand 24 fps as it's the default cinema one. You need to set your framerate according your assets. If you are using royalty free footage set a 29,97, your project framerate will need to be 29,97.

Tip: for the same content, a 24 fps video will render faster than a 29,97 video. Just because we will render 5,97 frames less per second.

CHAPTER 9

Using the SxC Validation Script

In order to help you make sure that your project is compatible with our engine, we've built an After Effects Script that will scan your project and report any error it encounters. it's name is: SxC Validator; SxC stands for *S*tupefli*X C*reative*.

9.1 Script Basics

Current version is 2.4 (Released October 29th 2014).

You can download it here for CS6 and above: Sxc Validation Script

To check if your project is good to go, open it in After Effects and then:

- File > Script > Run Script File (swith the dropdown from *.jsx to *.jsxbin)
- Select the script
- Select, in the Project panel, your widget main composition
- Clic on "Project Analysis".

The script now reports several key project information right inside it's UI: The number of compositions (plus the number of time they are used) and same goes for your layers and your placeholders. It also gives you your project depth, the highest layer stack and the number of errors/warnings it found.

If the script finds anything that is not complient with our workflow, or anything that could impact visualy the final look once it's in our system, it will generate a text log, with links to this documentation for each problem it found.

On top of these details, the script will give you an estimation of the Render Speed of the project on our render engine.

I highly recommand you to run it regularly while you are developing your widget to make sure you take the appropriate steps. Discovering at the end that you went the wrong way is always depressing and irritating.

9.1.1 WARNING

The script is just a tool built to help you investigate, find and correct the issues your project might have. Because of some scripting limitations, it cannot scan everything and thus, report everything. If your project scan is successfull, it means that there is a very high probability that it will be compatible with our system without further modification.

When receiving your AE projects, we will review them and will report you back asap if we find anything problematic that was not raised by the script.

Please note that the script covers around 95% of what we support, and whenever I find a situation that can be added to it, I'll update it. Be sure to come back regularly for new updated versions.

9.2 Scoring System

Starting with the version 2.0 of the script, we introduced a "scoring" system built to help you estimate how fast and how light your project will be when turned into a Stupeflix Widget.

In the text log header, you'll see a "Score" line looking like this:

```
Score: 0.59 (average per layer)
```

It's a score based on the comps & layers, the depth, the effects and everything you put in your project. We assigned a default score to every items, depending on how they perform or impact our rendering engine. We also take into consideration the complexity of your project for any given frame, and weight the score on it too.

The score found in the header as described above is your project total score, frame by frame divided by the total length of your Widget giving you an estimate of how time consuming it is. Here are the performences you can expect depending on your score.

As your project complexity can vary during time, we added a temporal complexity table, looking like this:

```
Time Stamp:          | 0    | 10   | 20   | 30   | 40   | 50   | 60   | 70   | 80   | 90
Layer Stack:         | 12   | 12   | 12   | 12   | 12   | 12   | 12   | 12   | 12   | 12
Project Score:       | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1
Death Combo:         | No   | No   | No   | No   | No   | No   | No   | No   | No   | No
```

The first line, **Time Stamp** is the frame number. We display show 10 frames, every 1/10 length of your project. The second line, **Layer Stack** is the height of the layer stack at that Time Stamp. The third line, **Project Score** is the complexity score of the project at this Time Stamp. The lower, the better. The last line, **Death Combo** is if what we call a *Death Combo* have been found at that Time Stamp.

9.2.1 Death Combo

Some things are time consuming in our engine, and when combined with other things, it can become really ugly.

We have identified 3 Death Combo, and here they are:

- Stack Death Combo: Staking more than 15 layers reduce rendering performance
- Placeholder Death Combo: showing too many user assets at the same time reduce rendering performance too
- trackmatte Death Combo: Nesting trakmattes, or having simultaneous trackmattes crush rendering performance. You really wanna avoid that.

9.3 Estimated Render Time

To give you an idea of how fast your project will render, we have captioned the score into 11 stages. And this time, you don't want to go up to 11.

- **ERS: Fastest**: Your project will be rendered faster than realtime. This is the best.
- **ERS: Very Fast**: Your project will be render slightly faster than realtime.
- **ERS: Fast**: Your project will be rendering more or less at realtime. That's the usual target.
- **ERS: Not so Fast**: Your project will be rendering in less than two time the realtime. Usually preview (half framerate) will be realtime.
- **ERS: Mediocre**: Your project is quite heavy and will take more than two times the realtime to render.
- **ERS: Sluggish**": Your project is heavy and will take around three times the realtime to render.

- **ERS: Slow**: Your project is really heavy and will take around four times the realtime to render.
- **ERS: Very Slow**: Your project is too complicated and will take around five times the realtime to render.
- **ERS: Ultra Slow**: Your project will take too long to render for our engine. We won't accept it.
- **ERS: Killing Slow**: Your project will take too long to render for our engine. We won't accept it.
- **ERS: Deadly Slow**: Your project won't be able to render on our engine. We won't accept it.

9.3.1 A grain of Salt

This scoring system is far from being bullet-proof, and your project might be faster or slower to render than what we actually expect. But from the test we've had in house with a large variety of projects, we are confident enough to make it available for you.

Also, for each composition in the text log generated by the script, you'll have a score line looking like this:

```
SCORE : 3.1 (total comp layer score)
```

This is the total score of your comp, which is just a sum of all it's layer scores. With this info you can see what compositions are render intensive and try to optimize them for better performance.

CHAPTER 10

Optimizing your Project

10.1 Naming Convention

We need your layers to avoid special characters in their name to be properly used within an XML context (changing layer color, hiding a layer. . . .).

Please, only use [a-z][A-Z][0-9] along with "_" in your layer names.

10.2 Pre-Rendering

Proxies and pre-rendering are really usefull when you have a bunch of layers with motion blur and effects that are not supported by our engine. They can also be used to reduce the amount of layers and effects to be processed by our engine, resulting in faster rendering times.

You can render your proxies and/or pre-renders in any file format we support. You can find a summary on our Supported File page.

Don't forget that you can also render semi-transparent animated footage, very usefull to add, for exemple, a layer of dust and scratches to simulate old film footage !

You can also use pre-rendered video in black and white to be used as a Luma or Luma Inverted track matte, especially if your track matte is a crowded composition with a lot of animations and effects such as blurs.

Pre-rendering a single image is quite straight forward: JPEG if it has no transparency, PNG otherwise.

Pre-rendering an animation is slightly more complicated as you have to keep the filesize low, and the compression low too, to keep a top quality level to avoid to much artifacting when it will be reencoded for the final user. For that we recommand using the h264 codec, 5.1 Level (High), vbr 2 pass encoding at 6mb target, 10mb max (for HD) or 3mb target 5mb max (for SD).

10.3 Start/End - In/Out Points

In After Effects, each layer have 2 starting points and 2 ending points. By default, they are the same, but sometimes you might need to change them, and they'll become out of sync. If that's not a problem for the ending / out point, our engine will struggle if the Starting / In point are different. But what are those points ?

Starting and Ending points are where the layer starts and end. In and Out points are where the layer is active.

The difference is subtle, here is an exemple: let's say you have a 5 seconds precomposition but want it to be trimed down to 3 seconds in your main comp. Either you'll shorten the precompo to 3 seconds by opening it and changing

it's duration in the composition setting panel, or you'll just trim it down in the main composition view by dragging the end of the layer with your mouse or your Wacom pen. In that second case, the Ending point will still be at 5 second, but the Out point will be at 3 second.

After Effects lets you trim your layer from the end, but also from the start of the layer, and it's clearly visible on precomp. If you move your comp layer in time the layer head won't change, but if you trim your comp layer head you'll see it become semi transparent. You can trim the head of any layer, but it will only be visible on comp layer. And it's easy, when you add a solid for exemple, to trim it to it's intended starting point instead of moving it. And you won't visually notice it. That's where it's complicated. So make sure to never trim a layer's head, but move it along the timeline.

It is mendatory that both Start and In point remains in sync. So if you need a layer to start at the 2 second mark in your timeline, don't trim its head, move the whole layer to it. Same if you need your layer to start a bit before it's actual position, don't expend it from it's head, move it entirely. Be sure to move your keyframes accordingly.

10.4 Assets Size

As you may have read in our Unsupported Features page, we don't allow graphical assets larger than 4096 pixel in width and/or 4096 pixel in height. That is also true for your composition sizes.

This limatation is due to graphic card memory and Open GL drivers to ensure best performances.

Still, 4096×4096 assets requires a lot of ressources and we encourage you to keep your project as fast as possible, to limit your assets size to 2048×2048.

Also, the smaller the assets file size, the faster it's processed, thus the faster the video is rendered. If you need transparency in your file, use PNG, if not use JPEG.

10.5 Output that Matters

We are now all used to work in native HD, may it be 720p or 1080p, and output at several range of resolutions, from SD for mobile devices and DVD to HD for broadcast and BluRay.

When creating a project for Stupeflix, you have to keep in mind that we are not rendering it once, we are rendering it everytime a customer calls it. So using oversized assets will slow down render time every time your assets is called.

The best practice is to create the AE project at the correct resolution to avoid as much as possible realtime downsizing during rendering.

For one of our client, we managed to divide by a factor of 2 the render time of their widget simply by reducing assets size, because they designed their project and rendered their assets in 1080p while they delivered the videos to their at a 360p resolution. This rendertime gain was only by reducing file sizes through assets resolution reduction.

To put it simple, if you are targetting to deliver 360p or 540p files to your customers through Stupeflix, the best practice is to deliver us AE projects in 360p or 540p.

If you plan to deliver multiple resolutions, you can either provide us a project at the highest resolution, or several projects to cover several resolutions.

Please note that just nesting your HD comp into a SD comp inside AE and scaling it to fit the new resolution is not the way to go because you are not scaling down your original assets, only the output, and you are instead adding 1 level of depth thus slowing down the whole rendering process.

10.6 Depth and Matte

The most render time intensive elements in an After Effects project are depth of the composition nesting, and the trackmattes. And the more you nest both, the bigger the render hit. That's what we call a Death Combo, and you can learn more about them here

Matting a precomp is something sometimes you can't avoid, but matting a precomp which already have matting inside will be killing the render time. If on top of that your precomps are exceeding the recommanded max size of 2048×2048 it will get ugly.

So beware of nested matting and try to avoid depth bigger than 3.

Also, if you need to stack several trackmatted layers, all using the same trackmatte, you will get better performance by precomposing your layer together and then only matting the precomp. This will save some render time

10.7 iOS Render Engine

If your campaign or your template is to be used on our iOS Renderer, you might take some more things into consideration:

- iOS Renderer supports bezier Keframes without rasterizing them.
- iOS Renderer will struggle with a layer stack higher than 10.
- iOS Renderer does not support video assets such as prerendered animated backgrounds.
- iOS Renderer won't accept assets larger than 2048x2048

CHAPTER 11

Further Optimizations

This section highlight some of the optimizations that are automaticaly suggested by our validation script based on your project analysis.

11.1 Merging Mattes

The script detected that you have several stacked mattes. This is going to severly impact the rendering time on our servers. One way to limit the overhead, is to create a video with an alpha channel on top of your assets, merging your background asset and your alpha mattes. That way the resulting alpha channel of the video will act as your cut out for your assets.

11.2 Nested Scale

We detected that some of your nested compositions are way bigger than their parent composition (ex: having a full HD comp nested in an SD comp). If you used nesting as a way of scaling the pre composition down, this won't make the rendering faster as we'll still need to process the full resolution. You should really try to lower the nested composition resolution as much as you can to improve rendering performance.

11.3 Duration Trim

We detected that some of your video assets are longer than the main composition. This affects performance because the bigger the file assets are, the longer they take to process.

11.4 File Size

We detected that your project is somehow heavy weigthed for its duration. You might want to reduce the file count or the assets bitrate. Bigger the file size are, the longer it takes our servers to process or to import. Big files can take some times even when they are streamed from one AMZ bucket to our. If render speed performance is key to your workflow, you might want to investigate this issue.

11.5 Video Stacks

We detected that you have 3 or more video stacked at some point in your widget. Decompressing and playing back several video streams at the same time can heavily slow down the rendering. You might want to merge them to lower the stream count. Remember that you can also render videos with an alpha channel using the On2VP codec in an FLV container.

11.6 Video UGC Continuity

When you plan to have video recording as UGC content, by default the video playback starts at the beggining of the UGC footage. But you might often want to spread the content of the UGC footage over several placeholders in time.

You'll need to create a placeholder for each slice of the video, and increment accordingly the placeholder index. Then, in the XML, you'll need to use the skip attribute of the video tag. The skip value for each placeholder is specified in the Placeholder section of the report.

CHAPTER 12

Using the Stupeflix XML Editor

The XML Editor is our home to test your XML definitions. An XML definition is usually a list of widgets, may they be from an After Effects project or a build-in Widget, and a list of assets to fill the placeholders.

If you've designed a video ad campaign, your XML definition will mainly be an ordered Widget succession.

But we can do much more. We use the XML definition to create stacks of widgets, sequences of Widgets, stacked sequences (yes, you can do both, bi-winning style !) transitions, and add audio.

As you see, there is much you can do with Widgets, but that's not all, we can even toy with the widgets themselves !

You can apply effects on the assets, modify the way they behave inside the placeholders, and even modify (show, hide, change color or opacity) layers from your After Effects Widget.

This ability to change build, combine, and modify behaviors directly inside the XML definition should be taken into account when you are designing your After Effects Widgets.

Not only we can reorder our bricks (to get back on that Lego bandwagon), but we can also change some paint to !

CHAPTER 13

Mixing your Project with XML

13.1 Applying XML filters

Filters are effects that you can apply on your assets and even on your Widgets.

The most used ones are:

- Alpha
- Colormap
- GaussianBlur
- Glow
- Kenburnsadapter

You can dig the full list here.

13.1.1 Basics

Before applying an effect, let's have a quick look at the heart of the XML defintion. Let's see that with a simple picture.

Watch and remix a copy of this project.

The XML Editor should open up, and the XML definition should look something like that:

```
<movie service="craftsman-1.0">
  <body>
    <effect type="none" duration="10.0">
      <image filename="http://stupeflix-assets.s3.amazonaws.com/help/assets/IMG_0024.JPG">
      </image>
    </effect>
  </body>
</movie>
```

As you can see, it doesn't look complicated at all. What have we here:

- A <movie> tag, very much like the <html> tag in web, we just define the kind of information we are going to have. We are making movies, so it's movie. Fair enough. We close the movie definition with the closing tag </movie>.
- A <body> tag, as in html, it's where we are going to put all the content of our video. The content will be defined between this tag and it's closing tag </body>.

- An <effect> tag. This tag defines how an asset should behave. The default type is "none", wich means we don't apply any transformation on any child. By child, I mean tags contained between <effect> and it's closing tag </effect>.
- An <image> tag. This is how we add an image asset to be used in our movie. The filename attribute defines the URL of the picture.

If you want to know more about the XML language we use, you can have a read at this page.

13.1.2 Exemple: Applying the colormap filter

So, we are going to add a filter to our picture. The filter we want to apply is considered a child of the picture, in the definition hierarchy. So the <filter> tag must be inserted within the boundaries of the <image> </image> tags of the picture we want to apply it to.

You should have something looking like that:

```
<movie service="craftsman-1.0">
  <body>
    <effect type="none" duration="10.0">
      <image filename="http://stupeflix-assets.s3.amazonaws.com/help/assets/IMG_0024.JPG">
        <filter>
        </filter>
      </image>
    </effect>
  </body>
</movie>
```

Now we have to define what kind of effect we want to apply. For this exemple I've choosen Colormap. It allows us to modify the look and feel of the picture.

```
<movie service="craftsman-1.0">
  <body>
    <effect type="none" duration="10.0">
      <image filename="http://stupeflix-assets.s3.amazonaws.com/help/assets/IMG_0024.JPG">
        <filter type="colormap">
        </filter>
      </image>
    </effect>
  </body>
</movie>
```

If you Generate the video, you'll see now that our picture is in black and white. It's the default behavior of this effect when neither a preset, nor a color matrix is called.

Now let's use a preset with our colormap effect. You can find the list of presets on the Colormap section of the filters page. I'll be toying with the "negative" one.

```
<filter type="colormap" preset="negative">
</filter>
```

If you generate the video, you'll see that the colors have been inverted, giving a negative look. See the result here.

If no preset gives you satisfaction, you can create your very own rgba-bias matrix. First of all, remove the preset attribute, at replace it with this funky one: rgba-bias-matrix

```
<filter type="colormap" rgba-bias-matrix="">
</filter>
```

Now we need to fill tha matrix with 5 hex values, one for each channel (red, green, blue, alpha), and a last one for the bias. Each hex value is composed of a # sign followed by 8 chars (a couple per channel, including alpha)

```
<filter type="colormap" rgba-bias-matrix="#FF000000,#00FF0000,#0000FF00,#000000FF,#00000000">
</filter>
```

If you generate the video, you'll see that our matrix changed nothing. That's because this matrix is the one our image uses by default. If you change the values of each channel, you'll see how it impacts the result. If your goal is just to give a color dominent tint to your picture, use the bias as it's it's job:

```
<filter type="colormap" rgba-bias-matrix="#FF000000,#00FF0000,#0000FF00,#000000FF,#aa000000">
</filter>
```

Red biased. See the result here.

So, as you can see, we can manipulate user assets data outside After Effects right inside the XML Definition. This allows us, to add effects that are not supported by our After Effects conversion process. Colormap is one of the mightiest one as soon as you understand how to play with the matrix system. It's far less intuitive than a curve system with a nice UI for exemple, but you (or your developer) can maniuplate and/or generate it on the fly while creating the definition. This way, you can make sure to have the results you want depending on the user assets color data. This of course requires some coding / processing on your application / website / program, but it's far more flexible than a simple AE Widget effect.

13.1.3 Exemple: Applying a filter on a Widget

Now that we have seen how to apply an effect on a user assets, let's see how we can do the same on this XML project.

It sould look a little like this:

```
<movie service="craftsman-1.0">
  <body>
    <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
      <text>What a wonderfull world !</text>
    </widget>
  </body>
</movie>
```

If we follow what we have done so far for the picture, we can imagine that adding the last colormap effect we have made would look like something like that:

```
<movie service="craftsman-1.0">
  <body>
    <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
      <text>What a wonderfull world !</text>
      <filter type="colormap" rgba-bias-matrix="#FF000000,#00FF0000,#0000FF00,#000000FF,#aa000000">
      </filter>
    </widget>
  </body>
</movie>
```

But if you generate the video, you'll see that nothing happens. That's because Widgets are a bit special, and the only children you can add, remove, change are the ones already built in it. In order to achieve the look we want, we have to embbed the widget in an higher-level object, and apply the filter on this object.

For that we are going to use the stack object, defined by the <stack> and </stack> tags. We are going to dig deeper into stacking and sequencing widgets on our dedicated page, but we'll introduce it here.

Let's start by encapsulating our widget into a stack. It should be somewhat similar to this:

```
<movie service="craftsman-1.0">
  <body>
    <stack>
      <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
        <text>What a wonderfull world !</text>
        <filter type="colormap" rgba-bias-matrix="#FF000000,#00FF0000,#0000FF00,#000000FF,#aa000000">
        </filter>
      </widget>
    </stack>
  </body>
</movie>
```

The filter is still our Widget's child. We have to move it into the stack like this:

```
<movie service="craftsman-1.0">
  <body>
    <stack>
      <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
        <text>What a wonderfull world !</text>
      </widget>
      <filter type="colormap" rgba-bias-matrix="#FF000000,#00FF0000,#0000FF00,#000000FF,#aa000000">
      </filter>
    </stack>
  </body>
</movie>
```

Now, if you generate the video, your widget will have the filter applied. You can see the result here !

13.2 Applying modifications to AE Widgets

With After Effects Widgets, we have introduced some features to give you slightly more control on how it will look:

You can hide a layer (any layer) You can change the color and the opacity of any solid layers Layers are obviously children of the Widgets, and so we are going to modify them directly within the <widget> and </widget> tags in our XML.

13.2.1 Hiding a layer

Let's have a look at this simple project. As you can see it's our Scrapbook Text Widget, nothing too fancy. If you remix it, the code should look like that:

```
<movie service="craftsman-1.0">
  <body>
      <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
        <text>What a wonderfull world !</text>
      </widget>
  </body>
</movie>
```

As you can see on the left and right side, there are some white spaces, which are in fact graphical elements of side picture borders, as you can add 2 user picture as a decoration. You can verify that right now:

```
<movie service="craftsman-1.0">
  <body>
      <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
        <text>What a wonderfull world !</text>
```

```
        <image filename="http://stupeflix-assets.s3.amazonaws.com/help/assets/IMG_0017.JPG" />
        <image filename="http://stupeflix-assets.s3.amazonaws.com/help/assets/IMG_0022.JPG" />
      </widget>
  </body>
</movie>
```

You may notice that the 2 pictures seems to miss their closing tag </picture>. As we are not applying any effect to them, we can use inline taging.

Take a closer look at the ends of those lines, and notice the / (slash) sign just before the > (greater than) :

```
.../help/assets/IMG_0017.JPG" />
```

This slash closes the tag. This is a handy tip to produce lighter and less bloated XML definitions.

Ok, so, if we don't want pictures, like in our first exemple, it looks like we are stuck with those white layers. Well not really. We can hide them. The only thing we need to know is their name.

That's where the Stupeflix Library comes into play. Because how would you know the names of the layers if you haven't yourself made the widgets ? On the Stupeflix Library, you have access to our list of Widgets, with code & render exemple, the list of accepted input data, and some information regarding the layers.

Let's have a look at our Widget Library page.

We can see that we have 2 layers pic01Landscape and pic01Portrait that are used as backdrops for our pictures. We can get rid of them. For that we use the <style> </style> tags. We need to use the attribute layer to define what layer we need to hide and the visible attribute to hide it.

Let's see it in action:

```
<movie service="craftsman-1.0">
  <body>
      <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
        <text>What a wonderfull world !</text>
        <style layer="main:pic01Portrait"  visible="false"/>
        <style layer="main:pic01Landscape"  visible="false"/>
      </widget>
  </body>
</movie>
```

You'll notice that i've also used the inline taging to save space on both <style /> lines.

Tadam, our 2 layers are now hidden. You can even remove the background map as you can see here.

13.2.2 Changing a solid color

Changing a color is as easy are removing layers, we'll also be using the <style /> tags.

First of all, let's have a look at this project. Remix it, it should look something like this:

```
<movie service="craftsman-1.0">
  <body>
    <widget type="set.moviestyle.machete.solo.01" duration="10">
      <image filename="http://stupeflix-assets.s3.amazonaws.com/help/assets/IMG_0017.JPG" />
      <text>Stupeflix</text>
    </widget>
  </body>
</movie>
```

This is a simple Widget, with a text and a user picture. To bake this extreme look, inspired by Robert Rodriguez "Machete" movie, the widget uses 4 colors:

- 2 for the background (radial ramp)
- 1 for the picture shadows
- 1 for the picture highlight

If we have a quick look at the Stupeflix Library, we get the 4 solid names. Now we just have to change their color in the <style /> tags.

```
<movie service="craftsman-1.0">
  <body>
    <widget type="set.moviestyle.machete.solo.01" duration="10">
      <image filename="http://stupeflix-assets.s3.amazonaws.com/help/assets/IMG_0017.JPG" />
      <text>Stupeflix</text>

      <style layer="background:lightColor" color="#F892FE"></style>
      <style layer="background:darkColor" color="#F600E2"></style>
      <style layer="pictureStyle:highlight" color="#F9B2FF"></style>
      <style layer="pictureStyle:Shadows" color="#0C000E"></style>

    </widget>
  </body>
</movie>
```

If you generate the video, you'll see that now we have nicely corporate pinkish colors.

As this Widget rely heavily on the user picture contrast, you can try to apply a colormap filter on the user asset to play with the contrast.

Here is the result with a lower contrast.

```
<movie service="craftsman-1.0">
  <body>
    <widget type="set.moviestyle.machete.solo.01" duration="10">
      <image filename="http://stupeflix-assets.s3.amazonaws.com/help/assets/IMG_0017.JPG">
        <filter type="colormap" preset="lowContrast" />
      </image>
      <text>Stupeflix</text>

      <style layer="background:lightColor" color="#F892FE"></style>
      <style layer="background:darkColor" color="#F600E2"></style>
      <style layer="pictureStyle:highlight" color="#F9B2FF"></style>
      <style layer="pictureStyle:Shadows" color="#0C000E"></style>

    </widget>
  </body>
</movie>
```

Notice how I turned back the picture inline tagging to a traditional tagging to allow the filter use.

13.3 Applying transitions

Until now we have seen how to modify a Widget inside an XML definition, and now it's tiime to see how to put several Widgets one after another, with and without transitions.

13.3.1 Putting two bricks together

Putting two Widgets together is as easy as putting them one after another in the <body> section of your XML definition. Let's have a closer look by remixing this project.

Should look mostly like that:

```
<movie service="craftsman-1.0">
  <body>
      <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
        <text>What a wonderfull world !</text>
      </widget>
      <widget type="set.scrapbook.map.01b" duration="10.0">
       <image type="map" center="Paris" zoom="15" />
      </widget>
  </body>
</movie>
```

You can see that we have just put 2 Widgets one after another to create this short video. But you can also see that in this case, the cut between the 2 scenes is not really eye candy. If you create a project on our studio with a simple text and a map, you'll notice that we apply what lloks like a slide transition between the scenes.

Transitions are used like Widgets and effects. You just need to encapsulate the type of transition you want inside the <transition> </transition> (or it's inline version <transition />) tags.

For the transition you will need to have at least 2 bricks as they only work between 2 objects.

Let's add the transition in our project:

```
<movie service="craftsman-1.0">
  <body>
      <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
        <text>What a wonderfull world !</text>
      </widget>
      <transition />
      <widget type="set.scrapbook.map.01b" duration="10.0">
        <image type="map" center="Paris" zoom="15" />
      </widget>
  </body>
</movie>
```

Now that our transition tag is properly inserted, we need to tell what transition we want. You have two choices. Either you choose from our list of built-in transitions, or you choose a custom made transition Widget.

For this first exemple, let's use the built-in transition move. We'll make this a 1 second left transition.

Should look like this:

```
<transition type="move" duration="1" direction="left"/>
```

or like this if you are using a custom made transition widget in After Effects:

```
<transition type="custom" duration="1" >
  <widget type="set.transitions.dissolve.01"/>
</transition>
```

If you look at the result here or here, you'll notice that the final duration of the video is 19 seconds and not 20 seconds as you might expect.

If we have a closer look at the XML defintion we clearly see that both widgets are 10 second long:

```
<movie service="craftsman-1.0">
  <body>
      <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
        <text>What a wonderfull world !</text>
      </widget>
      <transition type="move" duration="1" direction="left"/>
      <widget type="set.scrapbook.map.01b" duration="10.0">
        <image type="map" center="Paris" zoom="15" />
      </widget>
  </body>
</movie>
```

So why is the video 19 sec instead of 20 ? As the transition is an in-between state you might expect that a one second transition takes half second of the previous Widget and half a second of the next Widget.

Our system doesn'ty work that way, because during the length of the transition, we don't know how long both widgets are visible. For that reason, the transition "eats" it's full length on the ending of previous Widget and begining of next Widget.

For a 1 second transition, it acts as if you were overlapping the last and first second of the two Widgets you are transitioning. That's why our video is 1 second shorter than expected.

In the end, the duration of your video is reduced by the total duration of all your transitions. Duration calculations are then a bit difficult to grasp and calculate, but once you're used to it, it's a piece of cake !

13.4 Audio

Audio is not directly supported in our conversion process as widgets inside Stupeflix are mute. What you have to do, if you have audio in your project (like sound effects or music) is to render out a wav or an mp3 file of the audio and include it in your project.

We will then upload this audio file and link it to your widget inside the XML. If you want to have the audio file play at the same time of the widget, you'll need to stack them:

```
<movie service="craftsman-1.0">
  <body>
    <stack>
      <widget type="set.scrapbook.text.01b.m.01" duration="10.0">
        <text>What a wonderfull world !</text>
      </widget>
      <audio filename="url_of_your_file.mp3"/>
    </stack>
  </body>
</movie>
```