Hyle Documentation

This PDF version of Hyle's documentation is an excerpt of the online documentation, where you will find more detailed and recent documentation: http://hyle.io/docs

Introduction

Hyle is a powerful content generator for After Effects. It parses a simple and elegant syntax based on YAML.

Hyle will read through the file or text you feed it and generate content accordingly. Hyle is selection-aware, so the generated content will depend of your current selection (comp, layer, property, etc.).

Installation

Installing Hyle is just as staightforward as installing any other After Effects script.

1. Copy `Hyle.jsxbin` in your After Effects scripts folder.
    - Windows:
      `Program Files\Adobe\Adobe After Effects <version>\Support Files\Script UI Panel`
    - OS X:
      `/Applications/Adobe After Effects <version>/Scripts/ScriptUI Panels`
2. In After Effects general preferences check *Allow Scripts to Write Files and Access Network*

Using a good text editor

Although Hyle's UI provides a textarea, it is mainly for pasting purposes. Tabulations, which are very important when using Hyle, are not supported in the After Effects textarea. Therefore, it is strongly recommended to use an external

text editor such as Sublime Text (http://www.sublimetext.com/) or just a basic notepad app. Also, Hyle will support any kind of tabulations but the recommended settings are:

- Indentation style: space
- Indentation size: 2

A little bit about YAML

Yaml is a simple data-oriented language on which Hyle is built. You may have previously heard of XML or JSON as markup languages, YAML is just as powerful but is more readable and elegant.

Here's a basic YAML example:

```
- firstName: Tyler
  lastName: Durden

  professionalExperiences:
    - Soap Maker
    - Projectionist
    - Waiter

  favoriteQuote: >
    Without pain, without sacrifice, we would have nothing.
    Like the first monkey shot into space.
```

Read more about YAML: wikipedia.org/wiki/Yaml (http://en.wikipedia.org/wiki/Yaml)

Understanding items and layers

Items

Items are basically what you'll find in your **Project panel**. These are **Occurences** of objects, in opposition to **Instances** you'll find in your timelines.

That's why when you want to truly duplicate and modify a composition you have to duplicate it from the project panel. You then duplicate the **Occurence**.

Hyle's syntax has been created with ease of use in mind so you don't have to bother with After Effects different types of objects. But After Effects will create many things as items :

- Null, Adjustment and Solid Layers (that what's in that Solid folder you always get in your project structure)
- Compositions
- Folders
- Footage (footage refering to every type of media ex.: .mp4, .wav, .psd, etc.)

Layers

A layer is an item that has been staged. So when we create a solid layer in After Effects, two things happen. A new layer is **stored** in our project panel (in the Solids folder) and the layer is automatically **staged** in our active composition.

Object	Panel	Action	Type
Item	Project	Store	Occurence
Layer	Timelines	Stage	Instance

Items

The word item defines what you will find in your **project panel**.

Compositions

We will create a composition named "My first composition" with these settings:

- 1280 x 720
- Square pixels
- 10 seconds long
- 24 fps
- Black background

With this code...

```
compositions:
  - name: My first composition
    width: 1280
    height: 720
    pixelAspect: 1
    duration: 10
    framerate: 24
    color: 000000
```

In the example above, the composition will be **stored** in your project structure and be available in your project panel. Now what if you want to **stage** your composition as a layer inside of another composition?

```
compositions:
  - name: The master comp
  - name: The child comp
    parentComp: The master comp
```

As soon as the composition has a `parentComp` attribute, Hyle stages it inside of its parent.

The above example is really short and is missing most properties we stated previously. That's normal. There are default settings on which Hyle falls back if when a property isn't mentioned. Read more about defaults (/docs/defaults).

Folders

Folders were given a special shorthand syntax to simplify writing and improve elegance and readability.

```
folders:
  - folder 1
  - | folder 1.1
  - | | folder 1.1.1
  - | folder 1.2
  - folder 2
  - | folder 2.1
  - folder 3
```

Of course, the usual syntax will also work.

```
folders:
  - name: folder 1
  - name: folder 1.1
    parentFolder: folder 1
  - name: folder 1.1.1
    parentFolder: folder 1.1
  - name: folder 2
  - name: folder 2.1
    parentFolder: folder 2
  - name: folder 3
```

As seen above, `parentFolder` is used to specify the parent folder which value can be a name or an id (learn more about pointing to an item or layer (/docs/keeping-things-clean)).

Files

The keyword `files` will be used specify the start of a file array.

```
files:
  - path: /absolute/path/to/file.jpg
  - path: /path/to/another/file.mov
```

Layers

```
compositions:
  - name: The composition

layers:
  - name: The layer's name
    parentComp: The composition
    type: Solid
    width: 1280
    height: 720
    color: [0, 0, 0]
    duration: 10
```

Notice the `type` property. This is what will tell Hyle what kind of layer to create. There are 7 types of layers.

```
layers:
  - type: Solid
  - type: Text
  - type: Camera
  - type: Null
  - type: Adjustment
  - type: Text
  - type: Shape
```

As with compositions, the parent composition is pointed out with `parentComp` . Alternatively, you can use indentation to tell the parent composition.

```
compositions:
  - name: The composition
    layers:
      - name: The layer’s name
```

Blending Modes

The blending mode is available as a layer property. See the complete list of blending modes ().

```
layers:
  - name: The layer
    blending mode: multiply
```

Properties

Now creating composition and layers is cool, but we'll probably need more than that. Here's how to affect the different layers we've created.

Transform

```
layers:
  - name: The layer
    type: solid
    transform:
      scale: [50, 100]
      opacity: 50
```

That's it. Every transform property is available by using the name you see in After Effects.

Again, Hyle is selection-aware.

```
transform:
  position: [0, 0]
```

Will apply to all the currently selected layers in your interface.

Effects

```
effects:
  - type: Gaussian Blur
    name: My Gaussian Blur
    properties:
      bluriness: 50
```

The above snippet will apply gaussian blur and rename the effect "My Gaussian Blur" on the currently selected layer in your active composition.

Animation

Until now we have only used static values but Hyle will also support **keyframes** and **expressions**.

Keyframes

The `valueAtTime` keyword will create our keyframes. For each `valueAtTime` you have a keyframe will be created.

```
transform:
  opacity:
    valueAtTime:
      - time: 1
        value: 0
      - time: 5
        value: 100
```

Expressions

The `expression` keyword will specify expressions.

```
transform:
  scale:
    expression: time
```

We can use it while using a value and keyframes as well.

```
scale:
  valueAtTime:
    - time: 0
      value: [0, 100]
    - time: 1
  value: [100, 100]
  expression: loopOut("pingpong")
```

To use multiline expression, put the character `|` on the first line.

```
position:
  expression: |
    fps=5;
    amount=50;
    wiggle(fps,amount,octaves = 1, amp_mult = 0.5,(Math.round(
time*fps))/fps);
```

If your expression or if your expressions start with `#` or `[`, you will need to put your expression between simple quotes `'`.

```
position:
  expression: '[thisComp.layer("# Sequence").scale[0], 100]'
```

Keeping things clean

Now that's we've gone though the basics of outlining content for Hyle, you might realize that it would easily get messy with bigger and more complex files. Hyle has many tools under the hood to stay *DRY* and help us keep our outlines clean and short.

Using IDs

We have yet only referenced items and layers with their name and this can quickly get risky. IDs will fixe that.

```
layers:
  - name: Some Layer
    id: 1
  - name: Child layer
    parentLayer: 1
```

You can also give named IDs if you prefer.

```
layers:
  - name: Some Layer
    id: super special layer
  - name: Child layer
    parentLayer: super special layer
```

Overwriting defaults

Hyle has its own default values, here's how to overwrite them.

```
defaults:
  items:
    name: A default name
    layers:
      fontSize: 50
      transform:
        anchorPoint:
          expression: "[width/2, height/2]"
        position: [0, 0]
```

Inheritance

Inheritance will get handy when you'll need to create many items/layers without directly modifying the defaults.

```
layers:
  - type: Text
    id: 1
    text: Some text
    font: Helvetica Neue
    fontSize: 60
    transform:
      scale: [50, 50]
  - inherit: 1
    text: Different text
```

Fetching

Especially useful when nesting a composition in many places, fetching will allow you to simply add an item (comp, file, etc.) to another composition.

```
compositions:
  - name: The fetched comp
    id: 1
  - name: The main comp
    layers:
      - fetch: 1
```

Commands list

The language used here will assume you have read the preceding sections of the documentation.

Folder

Property	Type	Default
name	string	"Jase Default Name"
parentFolder	string	null
comment	string	null

Composition

Property	Type	Default
name	string	"Jase Default Name"
parentFolder	string	null
comment	string	null
selected	bool	false
label	string	null
id	int	null
width	int	1280
height	int	720
pixelAspect	int	1
duration	int	10

frameRate	int	24
bgColor	array	[0, 0, 0]
layers	object	null

Layer

Property	Type	Default
motionBlur	boolean	false
threeDLayer	boolean	false
adjustmentLayer	boolean	false
guideLayer	boolean	false
locked	boolean	false
blendingMode	string	"normal"
parentLayer	string	null
startTime	int	0
time	int	0
inPoint	int	0
outPoint	int	10

Blending Modes Available

add
alpha_add
classic_color_burn
classic_color_dodge
classic_difference

color
color_burn
color_dodge
dancing_dissolve
darken
darker_color
difference

dissolve
exclusion
hard_light
hard_mix
hue
lighten
lighter_color
linear_burn
linear_dodge
linear_light
luminescent_premul
luminosity

multiply
normal
overlay
pin_light
saturation
screen
silhouete_alpha
silhouette_luma
soft_light
stencil_alpha
stencil_luma
vivid_light

Text Layer

Property	Type	Default
fontSize	int	36
fillColor	array	[0, 0, 0]

strokeColor	array	[0, 0, 0]
strokeWidth	array	[0, 0, 0]
font	string	"Helvetica"
strokeOverFill	bool	false
applyStroke	bool	false
applyFill	bool	true
justification	string	"center"
tracking	int	0