

Expressions & Scripting

- Expressions
- Scripting

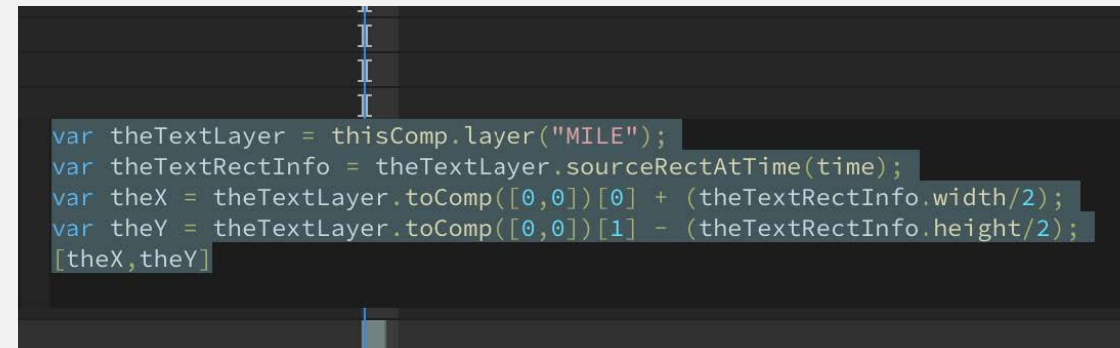
Expressions

After Effects **Expressions** are a code-based approach to keyframe evaluation. Rather than creating hundreds of keyframes and layers, expressions allow the user to code the value of a layer's property.

Expressions are **modular**, meaning that an expression written for one layer can be copied and modified to fit another layer easily.

Expressions can also define **relationships** between layers. One layer can inherit and manipulate the properties of another layer.

Expressions are written in **JavaScript** in the After Effects application.

A screenshot of the After Effects expression editor. The code is written in JavaScript and is used to position a text layer named 'MILE'. It calculates the center of the text layer's source rectangle relative to the composition's origin (0,0).

```
var theTextLayer = thisComp.layer("MILE");  
var theTextRectInfo = theTextLayer.sourceRectAtTime(time);  
var theX = theTextLayer.toComp([0,0])[0] + (theTextRectInfo.width/2);  
var theY = theTextLayer.toComp([0,0])[1] - (theTextRectInfo.height/2);  
[theX,theY]
```

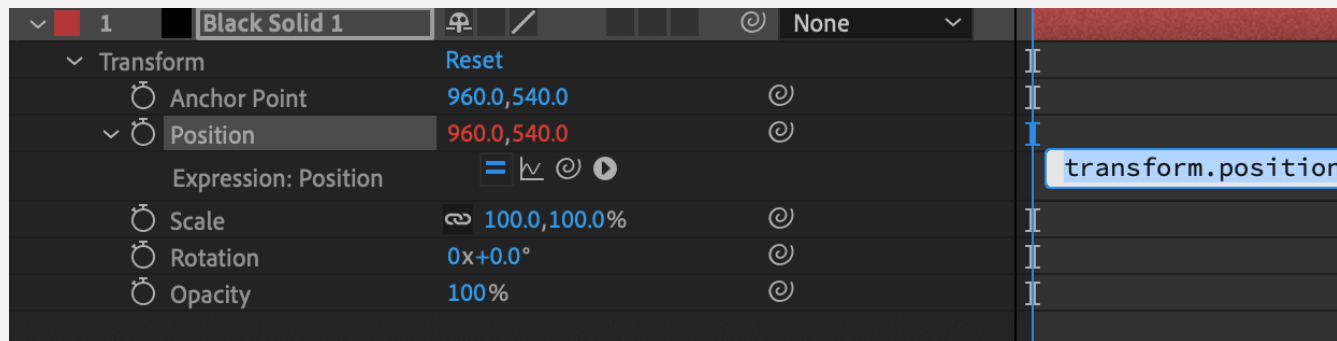
Expressions in After Effects

Creating An Expression

Alt/Option click on any property with a stopwatch to open up expressions for that parameter. The parameter's values will turn red, and a text box will appear to type the expression.

Expressions are **not strongly-typed**. The only restriction is that the return type must match the type taken in by the parameter.

For **single-value** parameters, it is alright to return a **single number**. For **multi-value** parameters, the expression must return an **array of values** depending on the number of parameters.



Enabling Expressions

Expression Key Terms

time returns a float of the composition time in seconds.

index returns an int of the index of the current layer in the composition.

numLayers returns the number of layers in the current composition.

width/height returns the width/height of the current composition.

duration returns the composition duration in seconds.

thisComp returns a reference to the current composition. Can be used to access other layers in the composition.

thisLayer returns a reference to the current layer.

thisProperty returns a reference to the current property.

value returns an int, float, or string of the selected property.

Expression Queries

layer('layer name') returns a reference to the layer with the provided name.

layer(index) returns a reference to the layer with the provided index.

transform.position returns the position value of the selected layer.

transform.rotation returns the rotation value of the selected layer.

transform.scale returns the scale value of the selected layer.

transform.opacity returns the opacity value of the selected layer.

Expression Interpolations

linear(t, start, end) returns a linearly-interpolated value between '*start*' and '*end*' at time '*t*' in range [0,1]

ease(t, start, end) returns a eased-in & eased-out value between '*start*' and '*end*' at time '*t*' in range [0,1]

easeIn(t, start, end) returns a eased-in value between '*start*' and '*end*' at time '*t*' in range [0,1]

easeOut(t, start, end) returns a eased-out value between '*start*' and '*end*' at time '*t*' in range [0,1]

For a comprehensive list, see <https://helpx.adobe.com/after-effects/using/expression-language-reference.html>

Using Expressions To Expand Animations

In an After Effects composition, create two ellipses on separate layers. For the first ellipse, open the scale parameter and generate keyframes with any magnitude and speed.

For the second ellipse, open up the scale parameter and alt/control click on the stopwatch to open up the expressions editor. Use the pick-whip to parent this property to the scale property of the first ellipse. The following should now show in the expressions editor:

```
(thisComp.layer("Shape Layer 1").transform.scale
```

The above expression directly inherits the scale keyframes from the first ellipse. We can make the animation start at the same value but expand twice as much using the following expression

```
(thisComp.layer("Shape Layer 1").transform.scale - [100,100]) * 2 + [100,100]
```


Using Expressions To Expand Animations

Unlike parenting that only allowed layers to parent onto one layer, expressions can inherit values from multiple layers. Given two different ellipses with different animation cycles, we can apply the following expression to a third ellipse:

```
thisComp.layer("Shape Layer 2").transform.scale - thisComp.layer("Shape Layer 1").transform.scale
```

The above expression is able to execute by returning a size-two array for the scale parameter that expects two values (x & y scale). transformation.scale could be replaced with transformation.position or hardcoded with size-two arrays and the expression would still execute.

Multi-Line Expressions

What does the following code do if applied to the scale parameter?

```
var t = time % 2;  
if(t > 1) {t = 2 - t;}  
var minX = 100;  
var maxX = 200;  
var x = ease(t, minX, maxX);  
var minY = 100;  
var maxY = 200;  
var y = ease(1 - t, minY, maxY);  
[x,y]
```

Answer: creates a squash-&-stretch animation cycle in 2 seconds.

- ~~Expressions~~

- Scripting

Uses of Scripting

After Effects Scripting extends Expressions by allowing users to **program composition & scene setup, layer creation, object interactions**, and more. Scripting is widely used in cases where artists have to repeat a routine multiple times and want to automate it or in cases when an animation is data driven and needs to parse the data.

Very common applications of scripting are in **motion capture parsing** where the script relies on a JSON of numbers specifying joint locations in order to render out primitives at the coordinates of these joint locations.

Scripting is useful for **data-driven animations** that require large amounts of data to be converted into keyframes. Such examples are trying to create simulations or visualizations from **sensor data**.

```
},  
"3817" : {  
  "0" : {  
    "left_ear" : [ 0, 0, 0 ],  
    "left_elbow" : [ 0, 0, 0 ],  
    "left_eye" : [ 0, 0, 0 ],  
    "left_foot" : [ 962.4450073242188, 351.2044677734375, 0.1523275375366211 ],  
    "left_hip" : [ 737.1214599609375, 71.02362823486328, 0.257271945476532 ],  
    "left_knee" : [ 852.722412109375, 217.9690093994141, 0.4809325039386749 ],  
    "left_shoulder" : [ 0, 0, 0 ],  
    "left_wrist" : [ 0, 0, 0 ],  
    "neck" : [ 0, 0, 0 ],  
    "nose" : [ 0, 0, 0 ],  
    "right_ear" : [ 0, 0, 0 ],  
    "right_elbow" : [ 0, 0, 0 ],  
    "right_eye" : [ 0, 0, 0 ],  
    "right_foot" : [ 0, 0, 0 ],  
    "right_hip" : [ 0, 0, 0 ],  
    "right_knee" : [ 0, 0, 0 ],  
    "right_shoulder" : [ 0, 0, 0 ],  
    "right_wrist" : [ 0, 0, 0 ]  
  },  
  "numPoses" : 1  
},  
}
```

Open-Pose Mocap Data

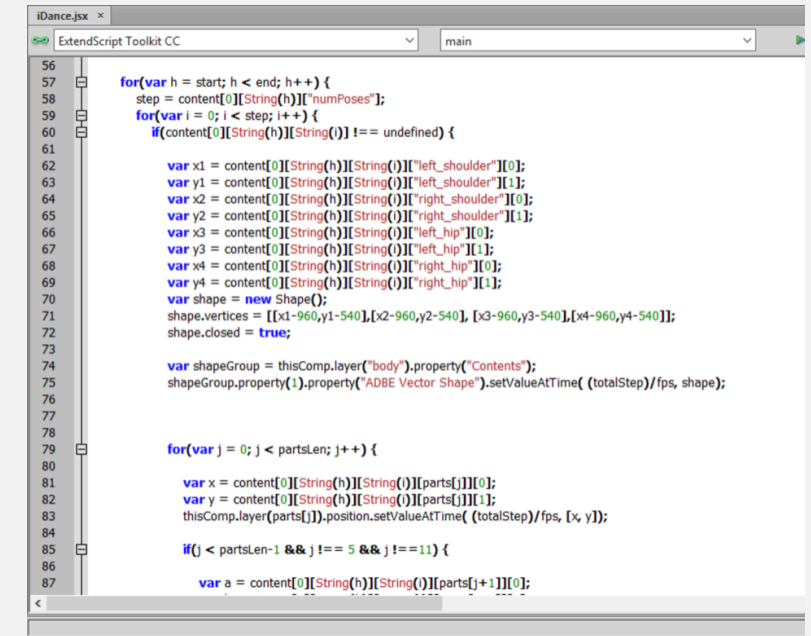
Scripting_Language

Scripting is written in **JSX**, a form of **JavaScript** that adds **XML syntax**.

XML is a language that allows users to write custom markup tags for rendering on the web. For the purpose of scripting, this extension is irrelevant.

JSX can be written in any IDE or text editor. None of what is coded in JSX has to be run or compiled down to any lower-level code. **After Effects** allows you to **open and execute JSX scripts** from the application itself.

Adobe also provides a text editor for writing JSX scripts configured with the Adobe scripting libraries for all CC softwares that offer library property completion, yet is not necessary to write scripts.

The image shows a screenshot of the 'iDance.jsx' script being edited in the 'ExtendScript Toolkit CC' application. The script is written in JavaScript with JSX syntax, using 'content' objects to access animation data. It includes a loop for 'numPoses' and another for 'partsLen', with various variable assignments and property access. The interface includes a toolbar with icons for opening, saving, and running scripts, and a dropdown menu for selecting the target application, currently set to 'main'.

```
56  
57  
58 for(var h = start; h < end; h++) {  
59   step = content[0][String(h)]["numPoses"];  
60   for(var i = 0; i < step; i++) {  
61     if(content[0][String(h)][String(i)] !== undefined) {  
62  
63       var x1 = content[0][String(h)][String(i)]["left_shoulder"][0];  
64       var y1 = content[0][String(h)][String(i)]["left_shoulder"][1];  
65       var x2 = content[0][String(h)][String(i)]["right_shoulder"][0];  
66       var y2 = content[0][String(h)][String(i)]["right_shoulder"][1];  
67       var x3 = content[0][String(h)][String(i)]["left_hip"][0];  
68       var y3 = content[0][String(h)][String(i)]["left_hip"][1];  
69       var x4 = content[0][String(h)][String(i)]["right_hip"][0];  
70       var y4 = content[0][String(h)][String(i)]["right_hip"][1];  
71       var shape = new Shape();  
72       shape.vertices = [[x1-960,y1-540],[x2-960,y2-540],[x3-960,y3-540],[x4-960,y4-540]];  
73       shape.closed = true;  
74  
75       var shapeGroup = thisComp.layer("body").property("Contents");  
76       shapeGroup.property(1).property("ADBE Vector Shape").setValueAtTime( (totalStep)/fps, shape);  
77  
78  
79 for(var j = 0; j < partsLen; j++) {  
80  
81   var x = content[0][String(h)][String(i)][parts[j]][0];  
82   var y = content[0][String(h)][String(i)][parts[j]][1];  
83   thisComp.layer(parts[j]).position.setValueAtTime( (totalStep)/fps, [x, y]);  
84  
85   if(j < partsLen-1 && j !== 5 && j !== 11) {  
86  
87     var a = content[0][String(h)][String(i)][parts[j+1]][0];
```

JSX Scripting Editor

Creating a Layer

Retrieve a reference to the active composition.

```
var thisComp = app.project.activeItem;
```

Create an empty shape layer and add a Vector Group. In the Contents of this vector group, add a new Vector Shape of specs Ellipse

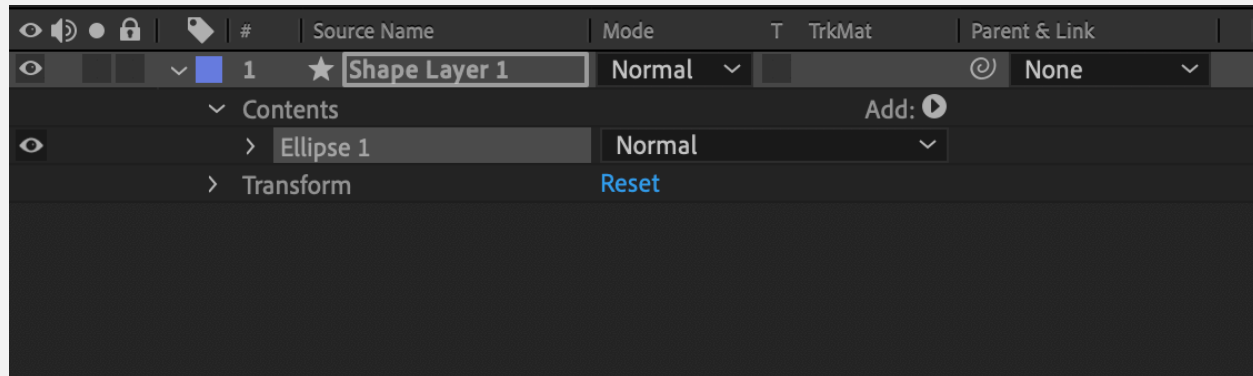
```
var shapeLayer = thisComp.layers.addShape();  
var shapeGroup = shapeLayer.property("Contents").addProperty("ADBE Vector Group");  
shapeGroup.property("Contents").addProperty("ADBE Vector Shape - Ellipse");
```

Creating a Layer

Why all this code to create an ellipse??

```
var shapeLayer = thisComp.layers.addShape();  
var shapeGroup = shapeLayer.property("Contents").addProperty("ADBE Vector Group");  
shapeGroup.property("Contents").addProperty("ADBE Vector Shape - Ellipse");
```

We first have to create 'Shape Layer 1', then let it know we are adding a new Vector Group, and in the 'Contents' property, we insert 'Ellipse 1'.



Keyframing

Extract the width and height of the composition. Query for the newly-created shape layer and set its position value at specific times.

```
var wth = thisComp.width;  
var hgt = thisComp.height;  
thisComp.layer('Shape Layer 1').position.setValueAtTime( 0 , [wth/2-100, hgt/2]);  
thisComp.layer('Shape Layer 1').position.setValueAtTime( 1 , [wth/2+100, hgt/2]);
```

setValueAtTime will automatically place keyframes at the layer's specified property. The first argument is a float indicating the time to insert the keyframe in seconds, while the second argument is the value at the keyframe. Here, position expects a 2D array of the x and y spatial values.

Scripting Documentation

Adobe keeps ~200 page documentation on the scripting library.

<http://blogs.adobe.com/wp-content/blogs.dir/48/files/2012/06/After-Effects-CS6-Scripting-Guide.pdf?file=2012/06/After-Effects-CS6-Scripting-Guide.pdf>

A cleaner version of the same documentation can be found here.

<http://docs.aenhancers.com>

Homework

- ☐ Continue working on Final Course Project.
- ☐ Take one or more concepts learned from this course to create your own animation. The expectation is that you have a rough storyboard or animatic of a few panels before moving on to the actual animation.
- ☐ You are allowed to incorporate found assets and footage so long as they are credited somewhere in the animation. You are also free to use any software, although help will only be provided for the list of software covered in this course.
- ☐ You should turn in a video file (audio is optional but appreciated) onto the course Drive.

Questions?

Live Demo