Exploring expressions and scripts in Adobe After Effects

Completed with the aim of graduating with a
Bachelor of Science in Engineering
From the St. Pölten University of Applied Sciences
Media Technology degree course

Completed by

Sebastian Mayrhuber
1110261086

Under the supervision of
DI Thomas Wagensommerer

Vienna, on

(Signature Author)

Declaration

The attached research paper is my own, original work undertaken in partial fulfillment of my degree.

I have made no use of sources, materials or assistance other than those, which have been openly and fully acknowledged in the text. If any part of another person's work has been quoted, this either appears in inverted commas or (if beyond a few lines) is indented.

Any direct quotation or source of ideas has been identified in the text by author, date, and page number(s) immediately after such an item, and full details are provided in a reference list at the end of the text.

I understand that any breach of the fair practice regulations may result in a mark of zero for this research paper and that it could also involve other repercussions.

Vienna, on

____
(Signature Author)

Abstract

This paper takes a close look at the two features 'expressions' and 'scripts' inside Adobe's compositing tool After Effects. The focus lies on exploring their purpose of use and finding out if they can be used to achieve results which couldn't actually be achieved without them. After a brief introduction to the topic, a detailed explanation, including real world examples, of the two features is given. A practical example near the end of the paper illustrates how to effectively combine expressions and scripts in one project and convert After Effects into a *Generative Art Composer*.

Table of Contents

List of Figures

List of Abbreviations

AE:	After Effects
API:	Application Programming Interface
CS5:	Creative Suite 5
HTML:	Hypertext Markup Language
Mac OS:	Macintosh Operating System
SDK:	Software Development Kit
VB Script:	Visual Basic Script
VBA:	Visual Basic for Applications
XML:	Extensible Markup Language
XMP:	Extensible Metadata Platform

1. Introduction

1.1. Problem and scope

This paper focuses on the often undervalued scripting possibilities of Adobe's composting tool *After Effects*. As for today, July 2013, After Effects reached version 12 (also called 'CC', short for Creative Cloud) and is one of the most popular layer-based compositing programs in the area of creating motion graphics and visual effects.

In today's industry the time factor is more important than ever. Tight deadlines and aggressive competition force professionals to work as quickly and efficiently as possible. The reason for this probably correlates with the ongoing rapid technological progress. Access to information and media is getting easier, digital media communication and therefore the need for time-based media content is constantly rising (Preissl, 2009, p. 53). In order to meet these requirements, *software* also has to constantly adapt itself to enable preferably smooth, flexible and quick workflows. After Effects and its user interface already does a good job in that regard compared to other software as it is rather clear and easy to understand. So there is not much room to improve on this side. But as tasks grow more complex, such as calculating positions of graphical elements in three-dimensional space or finding and updating source text in hundreds of text layers spread throughout the project file, a manual approach can quickly become rather tedious and time consuming. Fortunately, Adobe included the possibility to control its software and a lot of its properties via their own scripting languages 'Expressions' and 'Adobe ExtendScript'. Both languages are based on the popular 'JavaScript' language, which is rather easy to understand and learn. In that regard several questions arise: Which situations would it be wise to invest time in writing a script rather than doing a task manually? What are the boundaries of these 'features'? What can be achieved using scripts which otherwise would not even be possible?

1.2. Structure

The second part of this paper focuses on the concept of 'expressions' inside of After Effects. A detailed explanation of what they are, how and where they are written, and how they can be used inside the software shall be given. Furthermore, examples of popular expression commands shall demonstrate the real world usage of this feature.

The third part follows the second part's structure but revolves around 'scripts' in After Effects. A detailed explanation should point out the difference between scripts and expressions, how to automate repetitive tasks, perform complex calculations, and even use some functionality not directly exposed through the graphical user interface. Subsequently, examples of popular script commands shall demonstrate their real world usage.

In the fourth part a real world example employing both expressions and scripts is created and discussed. The goal is to demonstrate the thought processes which go behind the idea of using AE as a tool to create *Generative Art*. The script provides a user interface enabling users to customize the results and set parameters.

The last chapter serves as a summary and a critical reflection on the newly gained findings, looks at the actual boundaries of these techniques and points out an additional possibility to extend the functionality of After Effects.

2. Expressions in After Effects

2.1. Definition

Expressions are a powerful set of tools inside of After Effects which allow the user to control the behavior of layer properties and create complex relationships between them (Christiansen, 2011, p. 314). Layer properties are: position (defined by x and y coordinates), opacity, scale, rotation, applied effects and so on. Very similar to a programming script, an expression is a little software application defining a certain value of a layer property at a certain point in time (Adobe Systems Incorporated, 2012a, p. 616) – they are *typed commands* (Geduld, 2008, p. xi).

2.2. Purpose

In AE, you typically set keyframes for properties of your visual assets (images, videos, text, et cetera) which have been arranged on a composition frame to create an animation. By using expressions, you are able to assign static values to all kinds of keyframe-able properties as well but this is only their most basic function. A variety of commands allow you to *link* properties to other properties (even from different layers) to produce cascading behaviors or results which would otherwise be very difficult, laborious or even impossible to achieve (Christiansen, 2011, p. 314). You could for instance define that when an image is scaled up, a text below the image should also grow in size. In AE this mechanism is called *parenting* – a function which is also available on its own inside of AE, without the need of expressions. But by using expressions this concept becomes much more powerful and versatile (Geduld, 2008, p. xiii). Complex commands include the possibility to use mathematical operations, introduce randomness, manipulate time, transform values from a two dimensional space into a 3D space and many more (Christiansen, 2011, p. 315).

2.3. Use

So how exactly do expression commands look like and how and where are they applied? In the world of programming the available commands and functions combined are known as languages. Just like spoken languages, programming languages have a predefined vocabulary and certain rules to follow. Back in 2001 Adobe released version 5 of AE which was the first one to include the After Effects expression language (Siegel, 2010, p. 13). It is based on a subset of JavaScript, a programming language, which is mainly used in web browsers for creating dynamic HTML. Being a subset, it only includes the core functionality of JavaScript without any web browser specific extensions. What Adobe added are commands such as “Comp” (for “Composition”), “Camera“, “Layer” et cetera which allow you to access most of the values inside of AE (Adobe Systems Incorporated, 2012a, p. 624).

To create an expression you Alt+click (on Windows. Opt+click on Mac OS) the little stopwatch symbol you can find to the left of a property in the timeline panel or, if the property belongs to an effect, in the effect controls panel.

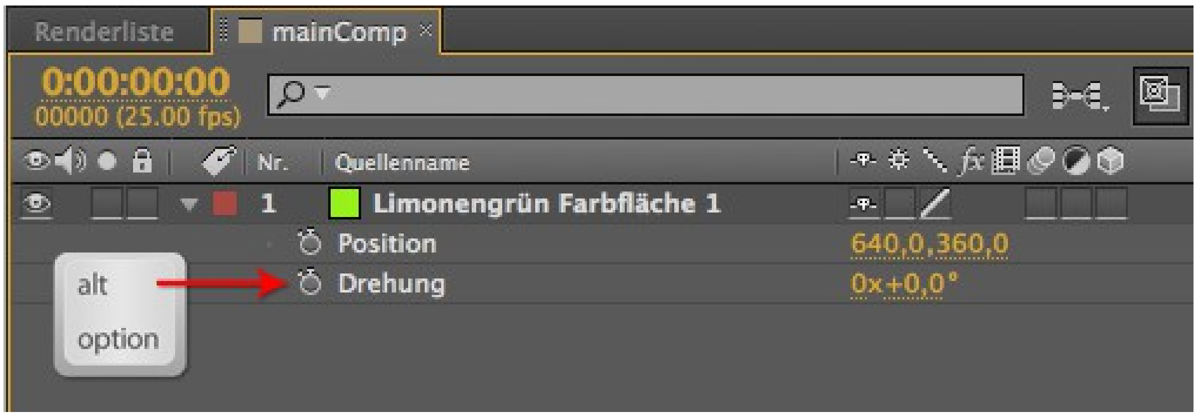


Figure 1: Create an expression using the stopwatch symbol. (Geduld, 2008, p. 3)

At this point, new buttons for controlling the expression and a text area to enter your commands appear. The button with the little right-faced triangle is particularly useful: clicking it reveals an expressions reference which lists all of the available commands and shows their correct syntax.

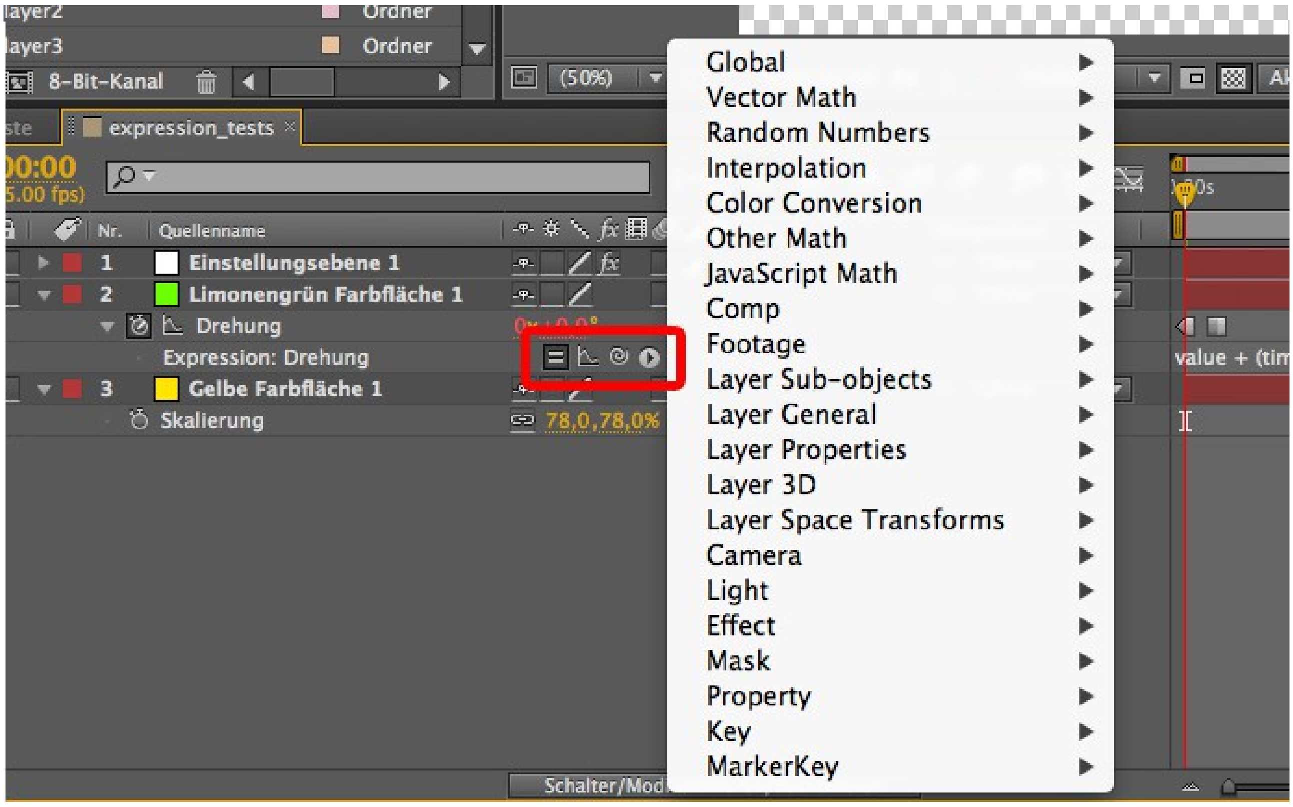


Figure 2: Buttons for controlling expressions with opened expression language menu.

The second button from the right is the *pick whip*. By clicking and dragging it onto another property or object AE will insert the correct expression text into the text area to link the two.

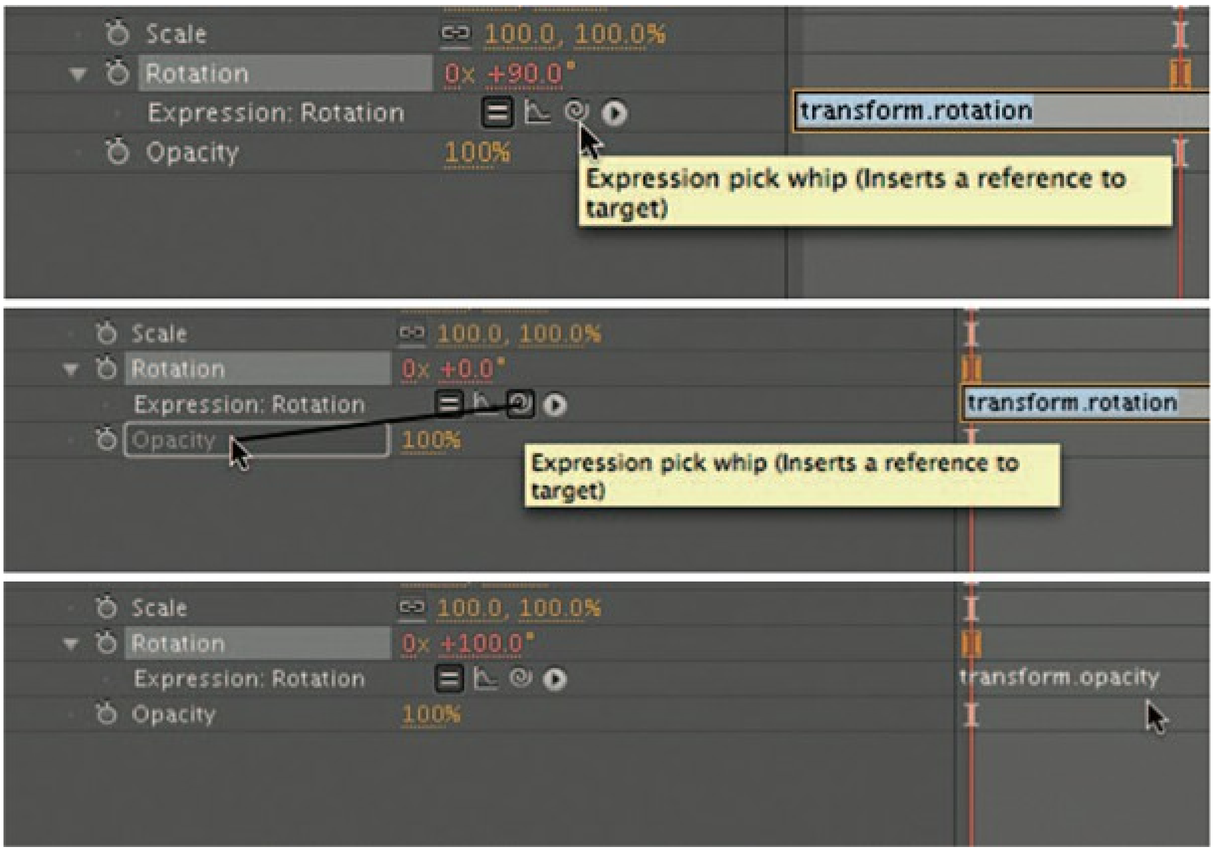


Figure 3: Using the pick whip tool. (Geduld, 2008, p. 22)

When using expressions, there are a few important details and limitations to be aware of:

> *An expression may generally be applied only to a property that can be keyframed, and it can affect only the value of that property. That is, an expression can affect one and only one thing: the value of the property to which it is applied. This means that although an expression has access to many composition and layer attributes (layer width and height, et cetera) as well as the values of other properties, it can only read, not change, them. (Christiansen, 2011, p. 315).*

In addition, if you have created an expression you are still able to set keyframes for that property. The expression gets *re-evaluated* once the keyframe is reached and it takes the new keyframe-value as the new *base* for the execution of the command (Adobe Systems Incorporated, 2012a, p. 616).

A convenient feature regarding expressions are *expression controls*: Basically, they allow you to setup a custom set of interface elements for controlling your expressions. You can choose between six different objects such as: slider, angle control or a color picker (Christiansen, 2011, p. 337). Attaching for example a slider control to a value inside of an expression enables you to update the otherwise static value more quickly and easily.

2.4. Examples

In the manual of AE you will find a complete reference with all the available commands listed and explained in detail (Adobe Systems Incorporated, 2012a, p. 629). It is very practical to, at least once, look through them all to get an idea of what is possible as most of them are very specific and will not be needed often. A few popular examples include:

The 'wiggle()' command:
The wiggle expression is quite fun to use as it randomly *shakes* the value of a property over time. How *wild* the shaking occurs can be controlled with two parameters passed on to the function. That means the expression `wiggle(5, 10);` results in 5 *wiggly* motions per second and each *wiggle* changes the value for about ten pixels, degrees, et cetera on average.

The 'random()' command-family:
This command returns random values. Different variations of this function allow you to restrict the resulting values to a certain range of numbers (e.g. random numbers between 50 and 150). This command opens up a lot of opportunities for personal imagination.

The 'loop()' command-family:
The loop command also exists in a number of variations. Basically, they all allow you to repeat a certain segment of a keyframed animation continuously. Through various parameters you can determine if the loop should jump to the beginning at the end of the animation (type = “cycle”), cycle back and forth (type = “pingpong”) and so on.

2.4.1. Example 'clock'

This short example of animating an analog clock illustrates how time-saving the use of expressions can be. The regular process of setting up rotation keyframes for the hour, minute and second hand throughout the length of your sequence would be quite tedious. This task can be done much easier and more efficiently if you use four simple lines of expressions.

Please note: This is not a complete guide for setting up an analog clock – only the most relevant parts concerning the topic of expressions are stated. The complete project file can be found on the CD attached to this paper.

1. First, the graphical assets are arranged on the composition frame and their anchor points are set to the center of the bottom edge so that the clock hands can rotate in the *correct* manner.
2. For this task the expression '`time`' comes in handy: `time` represents the composition time, in seconds, at which the expression is being evaluated (Adobe Systems Incorporated, 2012a, p. 630). With this information the current rotation angle of the respective clock hand can be calculated.
3. The second hand needs sixty seconds for one full turn (360°). That means in one second the second hand travels 360 / 60 degrees, which gives six degrees. When we add the following expression to the rotation property of the second hand, the hand will turn correctly timed for as long as the layer exists in the composition:
   transform.rotation = time*6;
   The result is a constant, 'fluid' motion as the expression is evaluated every frame of the determined movie frame rate (e.g. 25 frames per second). If you want the second hand to jump from one second to the next, like most analog clocks do, we can use another expression called '`posterizeTime(framesPerSecond)`'. This command lets us change the frame rate for a property to any value below the movie frame rate. So by adding the following expression before our previous expression, we can achieve the desired result:
   `posterizeTime(1);`
4. The minute hand follows the same principle: It needs 60 minutes for one full turn, that is 3600 seconds. That means in one second the minute hand travels 360 / 3600 degrees, which gives 0.1 degrees. We add the following expression to the rotation property of the minute hand to let it rotate at the correct speed:
   `transform.rotation = time*0.1;`
5. And finally the hour hand. One full turn takes 43200 seconds, so we add the following expression to the rotation property of the minute hand to let it rotate at the correct speed:
   `transform.rotation = time*0.008333333333;`

3. Scripts in After Effects

3.1. Definition

The process of developing a program run using a scripting language is called *scripting*. The resulting *script* consists of a series of commands which get executed one after another to complete a certain task (Siegel, 2010, p. 21). In contrast to the previously mentioned expressions which answer solely the purpose of reading and manipulating properties, scripts tell an application to *do* something (Adobe Systems Incorporated, 2012a, p. 616).

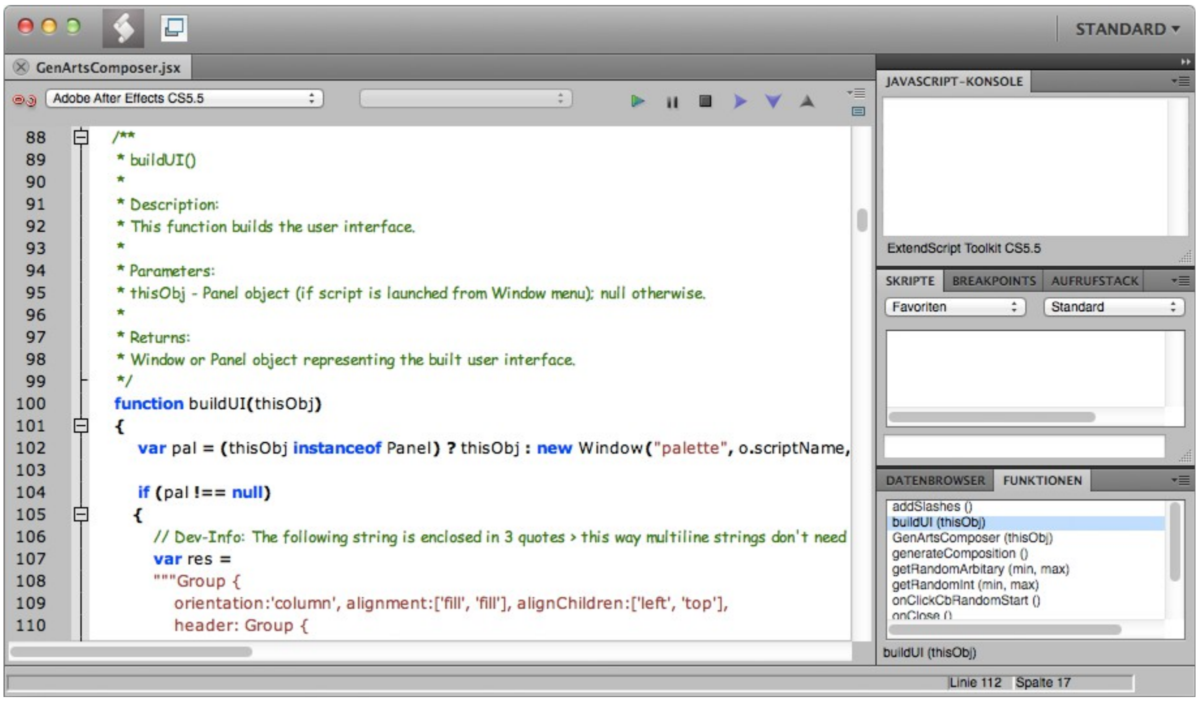
3.2. Purpose

The main reason Adobe included the feature of scripting into many of its tools is for automating reoccurring, time consuming, error-prone tasks such as resizing and saving hundreds of images. But the capabilities go much further: when it comes to After Effects, you can search for and edit source text inside of text layers, dynamically create objects such as compositions, layers and masks or send an e-mail as soon as the render process is complete, just to name a few (Adobe Systems Incorporated, 2012a, p. 613). With the possibility to read and write files (e.g. text files such as 'XML') and build custom user interfaces with the opportunity to configure individual settings, scripts are very well suited for being part of bigger workflows (Meyer & Meyer, 2010, bonus chapter 37c, p. 1).

3.3. Use

The topic of scripting around Adobe tools can be a little confusing at first because all scriptable tools (e.g. Photoshop, Illustrator, InDesign) support not only one scripting language but a few of them. Depending on your operating system, you can use AppleScript on Mac OS and VBScript, Visual Basic or VBA on Windows. JavaScript is supported on both platforms. The main scripting language though used to access After Effects functionality is called *After Effects ExtendScript.* Analogous to the aforementioned expression language it is also based on JavaScript and can be identified by the file extension *.jsx* or *.jsxbin*. The platform specific languages (AppleScript, Visual Basic, et cetera) can interact directly with some of Adobe's tools (Photoshop, Illustrator, et cetera) but in the case of AE they can only be used as kind of a *starting environment* to execute ExtendScript commands or scripts (Adobe Systems Incorporated, 2012b, p. 6).

Scripts can basically be written in any type of text editor but for enhanced comfort you can use Adobe's *ExtendScript Toolkit* which is included with all script-enabled applications. It represents a complete development and testing environment with features such as syntax-checking, running scripts from right within the tool without the need to save the file or a *debugger*, which allows you to inter alia execute scripts line by line which can be very helpful when e.g. trying to spot errors (Adobe Systems Incorporated, 2010, p. 13).



```
/**
 * buildUI()
 *
 * Description:
 * This function builds the user interface.
 *
 * Parameters:
 * thisObj - Panel object (if script is launched from Window menu); null otherwise.
 *
 * Returns:
 * Window or Panel object representing the built user interface.
 */
function buildUI(thisObj)
{
    var pal = (thisObj instanceof Panel) ? thisObj : new Window("palette", o.scriptName,

    if (pal !== null)
    {
      // Dev-Info: The following string is enclosed in 3 quotes > this way multiline strings don't need
      var res =
      """Group {
         orientation:'column', alignment:['fill', 'fill'], alignChildren:['left', 'top'],
         header: Group {
```

```
addSlashes ()
buildUI (thisObj)
GenArtsComposer (thisObj)
generateComposition ()
getRandomArbitary (min, max)
getRandomInt (min, max)
onClickCbRandomStart ()
onClose ()
```

Figure 4: 'ExtendScript Toolkit' editor.

There are several ways to run scripts. For example:

- You can copy your script into a specific scripts folder to get AE to load it when the application is started. The folder is located inside the AE installation directory. Inside of AE you can then run the script by navigating to the menu and choose: File > Scripts > [your script name].
- Choose and run a script by navigating to the menu and select: File > Scripts > Run Script File.
- If your script features user interface elements and is meant to appear in a dockable panel copy it into the 'ScriptUI Panels' folder which is located inside the AE installation directory inside the folder 'Scripts'. Inside of AE you can then run the script by navigating to the menu and choose: File > Window > [your script name] (at the bottom of the list).
- Scripts can be run from the command line or via AppleScript without having to manually

open the application at all. You can use this technique for example to execute a script when a custom keyboard shortcut is pressed (Adobe Systems Incorporated, 2012c, pp. 5–7).

According to Meyer and Meyer,

> *… scripting in After Effects is object based. From a script's perspective, After Effects appears as a hierarchy of objects. Each object in the hierarchy has associated methods that you can invoke and attributes that you can examine (and often alter). Think of objects as "things" (projects, comps, layers, cameras, masks, position property, et cetera). Think of methods as actions that objects can perform. Think of attributes as characteristics of objects that you can examine and sometimes modify. … When you write a script, you are essentially just navigating the object hierarchy, using the methods and attributes of the objects along the way to accomplish your objective. (2010, bonus chapter 37c, p. 2).*

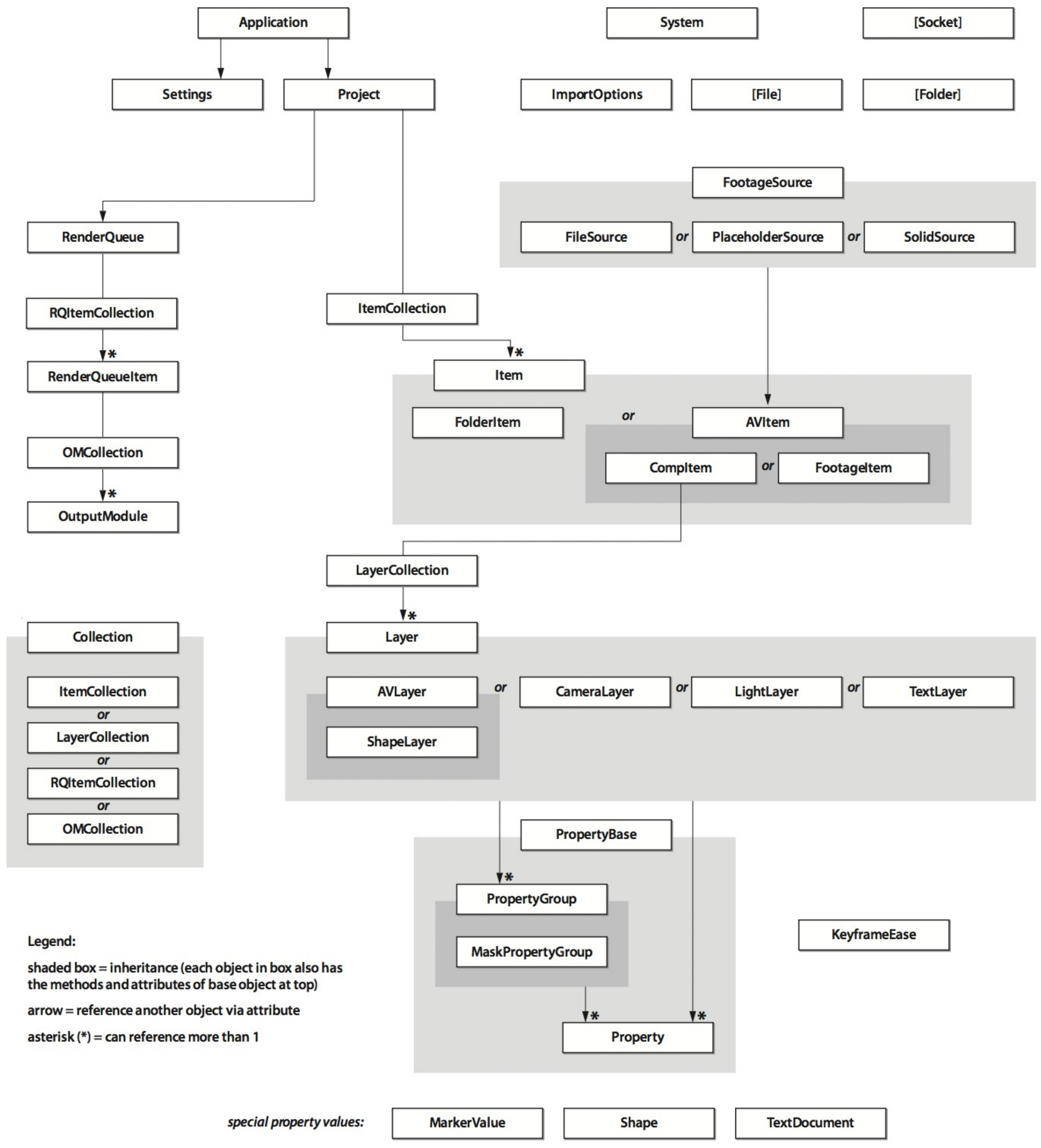


Figure 5: AE object hierarchy: This diagram shows the relationships among After Effects objects, how to access one object from another, and how some objects are based on other objects. Reprinted from Appendix 'Scripting' of Adobe After Effects CS5 visual effects and compositing studio techniques (p. 7), by M. Christiansen, 2011, Berkeley, CA: AdobePress. Copyright 2011 by Mark Christiansen.

Most of what can be done via After Effects' user interface can also be accomplished using the ExtendScript language, and the range of functions expands with each version (Adobe Systems Incorporated, 2012c, p. 3). But you have to keep in mind that the main purpose of scripting is to reduce the user's workload when faced with redundant, cumbersome tasks (Adobe Systems Incorporated, 2012a, p. 613). If you want to bring completely new functionality into AE you can do this by developing *plug-ins* using the programming languages *C* or *C++* in conjunction with the *After Effects SDK* (i.e. *Software Development Kit) (Adobe Systems Incorporated, 2012a, p. 611).*

3.4. Examples

For the field of scripting Adobe offers an independent, extensive manual called *Scripting Guide*. Inside, you will find detailed information about all the classes, objects, methods, attributes, and global functions defined by After Effects ExtendScript along with some examples. Again, like already mentioned in the examples section of the expression chapter, for being able to estimate the full potential of scripting inside of AE it is a good idea to at least skim through this guide once.

Common attributes include:
- “`app.project`” for accessing the current project object.
- “`app.project.item(1).name`” for getting the name of the first object in the project panel.

Common methods include:
- “`app.project.item(index)`” for retrieving an object from the project panel identified by its index number.
- “`app.project.item(index).layers.byName(name)`” for retrieving the first (topmost) layer object with the specified name inside of an item object with the specified index number.

3.4.1. Example 'SetGuideLayers'

This short example demonstrates a possible scenario in which the use of scripts comes in handy. AE offers the possibility to turn any layer type, except cameras and lights, inside of a composition into a 'guide layer'. Guide layers are inherently not included in the final render process so they can be used as a reference while for example arranging elements on the screen. The following script turns all selected layers which include the word '*guide*' in their names into guide layers. In all other cases the guide layer attribute is set to *false*. By following this simple naming convention, a user can save time before launching the render process. Instead of manually editing all the relevant layers within the composition he just selects them all, launches this script and all his *guide* layers will be hidden in the final rendering.

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

for (var i = 0; i < comp.selectedLayers.length; i++)
{
    var layer = comp.selectedLayers[i];

    var setGuide = layer.name.indexOf("guide");

    if (setGuide == -1)
        layer.guideLayer = false;
    else
        layer.guideLayer = true;
}
```

4. Case study: 'Generative Art Composer'

4.1. Introduction

This chapter describes the development process of a generic project in which both of the two aforementioned scripting tools are deployed. A case scenario was created which could not be realized without the help of AE's scripting abilities, at least not to the same extent. Through close examination of the subject matter, the author gained valuable knowledge in the course of being able to answer this paper's research questions.

The basic idea behind this script seizes a contemporary form of artistic pursuit: Generative Art. As Florian Josef Gruber states,

> *… Generative Design can be defined as a digital form of art, whose shape and process is not determined in detail, is constantly evolving and up to a certain degree random and unforeseeable. One has to develop processes that for one intend a certain aesthetic policy and organizational principle (theme) and additionally implement a circular system for aleatoric change (variation). (2012, p. 249).*

After setting up this framework the process of creating new work is then executed by a computer software, media or other aids. The focus is not so much on the resulting artwork but rather on process of formation itself and the ideas behind it ("Generative Kunst," 2013).

This script brings this concept into After Effects. The user provides general guidelines but it is the tool itself which is then left with the task to create a video based on random values and user generated content. Random elements are chosen out of a pool of assets and placed on the timeline, be them single images, image sequences, video clips or audio clips. Also an option is provided to let the content of the chosen items start from random points in time and repeat in a loop once they ended. This way every execution of the script results in a new, unique composition of graphical elements. After Effects gets misused – rendering it a tool to create 'Generative Art'.

4.2. Script process

Before going into detail, the author likes to point out that basic programming knowledge is necessary to understand the following explanations, in particular the syntax of the JavaScript language, as the goal of this is to give the reader an idea about how the procedure of writing a script for Adobe After Effects can look like and which issues one might encounter. Also important to note is that, for the sake of this paper, it was not considered necessary to reach a *production-ready* status meaning that a *correct* application of the script is expected and not *all* possible sources of errors are being caught.

In the beginning a few rules have been defined for this script to work. AE has to be launched and a prepared project file has to be opened before the script can execute. The user has to provide modules out of which the final video will be composed. That could be prepared AE-compositions, movie files, image files – really anything AE allows to import and place on the timeline. The assets have to be placed inside folders which follow this simple naming convention: “layer1”, “layer2” and so on. The reason behind this lies in the scripts' ability to stack assets on top of each other. Assets placed inside 'layer2' will appear above assets from 'layer1', assets from 'layer3' above assets from 'layer2' and so on. The number of folders is not limited. The elements themselves can be named arbitrarily.

The script is divided into four areas: 'Init Vars', 'Functions', 'Main' and 'Helper Functions'. For the purpose of a better understanding of the whole procedure the areas will not necessarily be explained in that order but the author will describe the process in the order of how the commands get executed by the application. First, an object 'o' is declared on the root-level of the script as a container to hold all the variables which should be globally accessible. When debugging with the ExtendScript Toolkit the 'Data Browser' panel can be used to examine and set variables and a big advantage of using such a global object is that all your important variables are then clearly displayed there in one spot and not scattered throughout the panel. The global object also holds translation-string variables for every text displayed in the user interface as the script is already prepared to be displayed in various languages by using the built in global function '`localize()`'.

Then the 'Main'-area is processed beginning with an evaluation of AE' version number. 'CS5' (= v10.0) or higher is required to execute this script. The current version number is stored in the property '`version`' of the global object '`app`'. '`app`' is essential as it defines the root of the object hierarchy from which most of the scripting functionalities are accessed. Following next is the procedure of building the graphical interface for receiving user input. AE offers several ways to consolidate interface elements onto *containers.* This script supports two variants: Depending on how the script is launched it either shows up as a *floating palette* or as a *dockable panel* with the option to get launched automatically when AE starts (see chapter 3.3). Then all the visible elements are created: 'EditText' objects to let users input text and determine the features of the video composition to be generated, 'CheckBox' objects for activating or deactivating certain functions of the script, 'Button' objects to initiate displaying a help message and for launching the generation process and 'StaticText' objects for labeling all those interactive elements. Through the help of the 'Bounds'-, the 'Dimension'- and other objects, AE gives you the option to explicitly define x- and y-position- and width- and height values for all elements. But in this case the 'automatic layout' capability of ExtendScripts' ScriptUI component combined with the technique of using 'resource specification strings' was chosen. This way proper sizes and positions for control elements and containers are automatically determined with the additional option to manually intervene and define for example if the child elements of a container should be organized in rows or columns, where they should be aligned within the available space or how many pixels child elements should be separated from each other ('*spacing*') (Adobe Systems Incorporated, 2010, pp. 86–91). The 'buildUI' - function ends by setting up event-handlers that define which actions are carried out when the user interacts with certain elements. The remaining part of the main area deals with loading and saving the current state of the user interface. This handy feature of AE allows for storing custom defined 'key and value'-pairs in a preference file enabling you to bring over data from one application session to the next.

By clicking the “Generate Composition”-Button in the bottom-right corner, the prime function of the script '`generateComposition()`' is called. In its first lines the available input textfields are evaluated. With the help of those a user can set the essential properties of the container composition the script is about to create. Like already mentioned in the beginning of this chapter, only a simple 'filled in or not'-query is implemented at this point. For a fail-safe execution of the script additional queries such as ensuring that the input values reside within controlled ranges, would have to be implemented. Next, the evaluation of the first setting is carried out: the user is given the opportunity to let the script automatically append a 'timestamp' at the end of the composition name. A very useful option as After Effects allows multiple elements in the project to have the same name. Using this setting unique names are generated each time the script is executed and the new composition can be identified more easily. In the following step the project is searched for folders which match the aforementioned naming convention. References to the found folders are stored in an array for easier access later on. Afterwards the container composition is created on the basis of the user's input and is also immediately *selected* in the project panel for easier spotting. Now the essential part of the script takes place: The random addition of items into the timeline of the container composition. The process starts with the folder 'layer1' and works its way *upwards* until it reaches the last folder. With the help of the function '`Math.random()`' an arbitrarily item is chosen out of the folder and added to the timeline. At this point two further user settings come into effect which help generate an even *more random* end result: The user can decide wether the items themselves should begin at random points in time and if so, if their playback should loop once they reached their final frames and play on for as long as they originally lasted or not. This is achieved by using ExtendScripts' ability to dynamically add expressions to item properties. In this case the '`timeRemap`' property is driven by an expression which offsets its value by a random number every frame which results in the desired effect. For single frame items which have no duration per se these settings are ignored and they simply get assigned a random duration between one and five seconds. After the item has been successfully added to the timeline the script stores the current duration of the main composition in a helper variable '`curLength`' in order to know when the next random item should start. This process of adding items is now repeated until the defined composition length is reached. The whole process is completed once all found 'layer' folders are processed. At last, the final composition automatically opens for a quick and comfortable review if the user chose so in the settings.

5. Discussion

With the integration of expressions and the ExtendScript language Adobe managed to provide a fascinating and versatile extension to their popular compositing tool. They are not exactly prominently placed inside of AE' user interface, waiting for the more experienced user to be found and explored. Through the course of researching and experimenting on the topic of this paper the author gained deep insight into the purpose and the capabilities of these tools. The main idea behind both features is certainly to act as *servants* helping you to reach your aimed goals in less time. Whenever you find yourself doing certain tasks over and over again, doing something which follows an exact procedure or, in the case of expressions, want a layer property to behave in certain way or be *linked* to *any* other property *anywhere* in the project, it might be a good moment to take a look into the well edited reference documentations provided by Adobe (see reference section). Depending of course on the programming skills of the person, getting used to Adobe's programming language is not that difficult. The JavaScript-syntax is comparatively accessible and easy to learn.

Expressions and scripts really expand the functionality of AE in many ways and enables the user to do things which could not be achieved otherwise. The expression language includes many special functions allowing you to, for example, calculate physical simulations (using sine waves, Euler's number, the natural logarithm and much more), sample color data from images or easily setup custom made interpolations between different sets of values. Granted, nearly all of these *scripted* values could also be calculated elsewhere and then set by hand but the result and the time taken to get there would be completely out of proportion.

ExtendScript scripts present a similar picture. This paper focused on the After Effects-specific set of functions which make nearly every aspect of AE controllable. But considering the entire range of functions the ExtendScript language offers the overall capabilities are greatly enhanced. Here is a small excerpt:

- The method '`system.callSystem()`' allows executing system commands and process potential return values as if you would type them on the operating system's command line.
- An application programming interface (API) enables communication between all scriptable Adobe tools.
- External communication is enabled through the *Socket* object. This way network connections can be setup up allowing you for instance to exchange data with internet servers.
- The programming environment can be extended by loading external *shared libraries*

(written in C or C++) through the *ExternalObject* object.

- Possibility to process XML documents by default.
- Possibility to access XMP standardized metadata.

All these modules combined make up for a pretty limitless use. And that this abundance of possibilities is actually employed is easily comprehensible by looking for example at the constantly growing range of scripts on websites such as www.aescripts.com.

In the authors opinion the option to introduce randomness to your project alone is worth the time invested in learning these techniques as this greatly enlarges the creative scope. This set of functions will most likely lead you to develop more flexible solutions and consult After Effects for tasks which you did not think of before. And should you hit any boundaries, Adobe has another solution ready which is even used for the all the native effects inside of AE: the *plug-in* architecture. Plug-ins are little software modules developed with, among other things, the programming languages *C* or *C++* using Adobe's *Software Development Kit (SDK)* and make it possible to introduce completely customized functionality to After Effects.

6. Appendix

6.1. References

6.1.1. Research papers

Gruber, F. J. (2012). *If Code == Imagination : Theorie, Modelle und Entwurfsmuster generativer Designprozesse*.

Preissl, G. (2009). *Gestaltungsfaktoren im Motion Grafic Design*.

Siegel, M. (2010). *Workflowoptimierung mit After Effects (am Beispiel c-tv)*.

6.1.2. Books

Christiansen, M. (2011). *Adobe After Effects CS5 visual effects and compositing studio techniques*. Berkeley, CA: AdobePress.

Geduld, M. (2008). *After Effects Expressions*. Amsterdam; Boston: Focal Press/Elsevier.

Meyer, T., & Meyer, C. (2010). *Creating motion graphics with After Effects: Version CS5*. Amsterdam: Focal Press/Elsevier.

6.1.3. Manuals

Adobe Systems Incorporated. (2010). *Adobe Creative Suite 5 - JavaScript Tools Guide*. Retrieved June 23, 2013, from http://www.adobe.com/content/dam/Adobe/en/products/indesign /pdfs/JavaScriptToolsGuide_CS5.pdf

Adobe Systems Incorporated. (2012a). *Adobe After Effects - Hilfe und Übungen*. Retrieved June 15, 2013, from http://help.adobe.com/archive/de/after-effects/cs6/after_effects_reference.pdf

Adobe Systems Incorporated. (2012b). *Adobe - Skripte Einführung*. Retrieved from local installation directory of the Adobe Creative Suite CS6. On Microsoft Windows it can be found here: c:\Program Files (x86)\Adobe\Adobe Utilities - CS6\ExtendScript Toolkit CS6\SDK\German\Adobe Intro to Scripting.pdf. And on Mac OS here: ~/Applications/Utilities/Adobe Utilities – CS6 /ExtendScript Toolkit CS6/SDK/German/Adobe Intro to Scripting.pdf

Adobe Systems Incorporated. (2012c). *Adobe After Effects CS6 Scripting Guide*. Retrieved June 23, 2013, from http://blogs.adobe.com/aftereffects/files/2012/06/After-Effects-CS6-Scripting-Guide.pdf

6.1.4. Web pages

Generative Kunst. In **Wikipedia**. Retrieved September 13, 2013, from http://de.wikipedia.org/w/index.php?title=Generative_Kunst

6.2. CD content

- > root
  - > After Effects files
    - Expressions and scripts CS5.5 - Mayrhuber Sebastian BMT11.aep
  - > ExtendScript scripts
    - ConvertToGuideLayers.jsx
    - GenArtsComposer.jsx
  - > References
    - > Websites
      - Generative Kunst – Wikipedia.pdf

6.3. Script listings

6.3.1. ConvertToGuideLayers.jsx

```
/**
* ConvertToGuideLayers.jsx
* Copyright (c) 2013 Sebastian Mayrhuber. All rights reserved.
* portfolio: www.sebastianmayrhuber.at
*
* Name: ConvertToGuideLayers
* Version: 1.0
*
* Description:
* This script converts selected layers into guide layers if their names contain the word
'guide'.
* If 'guide' cannot be found in the name the 'guide layer' attribute of the layer is
unset.
*
* Usage:
* 1. Make sure every layer you want to turn into a guide layer has the word 'guide' in
its name.
* 2. Select one or more layers.
* 3. Run this script.
*
* Legal Notices:
* This script is provided "as is," without warranty of any kind, expressed or implied.
* In no event shall the script's author be held liable for any damages arising in any
* way from the use of this script.
*/

{
    var comp = app.project.activeItem;

    for (var i = 0; i < comp.selectedLayers.length; i++)
    {
        var layer = comp.selectedLayers[i];

        var setGuide = layer.name.indexOf("guide");

        if (setGuide == -1)
            layer.guideLayer = false;
        else
            layer.guideLayer = true;
     }
}
```

6.3.2. GenArtsComposer.jsx

```
/**
* GenArtsComposer.jsx
* Copyright (c) 2013 Sebastian Mayrhuber. All rights reserved.
* portfolio: www.sebastianmayrhuber.at
*
* Name: GenArtsComposer
* Version: 1.0
*
* Description:
* ..see variable 'o.strHelpText' below
*
* Usage:
* ..see variable 'o.strHelpText' below
*
* Legal Notices:
* This script is provided "as is," without warranty of any kind, expressed or implied.
* In no event shall the script's author be held liable for any damages arising in any
* way from the use of this script.
*/

/**
 * GenArtsComposer()
 *
 * Description:
 * This function contains the main logic for this script.
 *
 * Parameters:
 * thisObj - "this" object.
 *
 * Returns:
 * Nothing.
 */
(function GenArtsComposer(thisObj)
{
    //
    // HELPER FUNCTIONS
    //

    /**
     * Returns a random number between min and max.
     */
    function getRandomArbitary (min, max)
    {
        return Math.random() * (max - min) + min;
    }

    /**
     * Returns a random integer between min and max.
     * This results in a uniform distribution! Using Math.round() would result in a non-
uniform distribution ->
     * http://stackoverflow.com/questions/1527803/generating-random-numbers-in-
javascript-in-a-specific-range
     */
    function getRandomInt (min, max)
    {
        return Math.floor(Math.random() * (max - min + 1)) + min;
    }

    /**
    * Returns escaped string (Quote, Double-Quote)
    */
    String.prototype.addSlashes = function()
    {
       return this.replace(/[\\"']/g, '\\$&').replace(/  0000/g, '\\0');
    }

    /**
     * openCompPanel()
     *
     * Description:
     * Opens a given composition. WARNING: To achieve this effect the undocumented
function 'workAreaDuration()' is used.
     * Don't use the according setting if you run into any errors.
     *
     * Credit goes to Rich Helvey on the creative-cow-forums:
     * http://forums.creativecow.net/thread/227/10710
     *
     * Parameters:
     * thisComp - The composition to be opened.
     *
     * Returns:
     * Nothing.
     */
    function openCompPanel(thisComp)
    {
        var duration = thisComp.workAreaDuration;
        thisComp.workAreaDuration = 0.06;
        thisComp.ramPreviewTest("",1,"");
        thisComp.workAreaDuration = duration;
    }


    //
    // INIT VARS
    //

    // Global Var-Object to be accessed from everywhere
    var o = new Object();
    o.scriptName = "GenArtsComposer";
    o.scriptTitle = o.scriptName + " v1.0";
    o.clipFolders = new Array();   // Store the references to the folders that hold all
the clips in an array
    o.sm_gacPal;  // Store reference to the main panel / window-palette

    o.mainComp;  // Reference to the main-comp which holds all the clips
    // Define default-values for UI:
    o.mainCompName = "GenArts_MainComp";
    o.mainCompWidth = 1280;
    o.mainCompHeight = 720;
    o.mainCompPixelAspect = 1;
    o.mainCompDuration = 20;
    o.mainCompFramerate = 25;
    o.cbTimestamp = true;
    o.cbRandomStart = true;
    o.cbLoopItems = false;
    o.cbOpenComp = false;

    o.strMinAE100 = {en: "This script requires Adobe After Effects CS5 or later."};
    o.strHelp = {en: "?"};
    o.strConfigureFinalComposition = {en: "Configure final composition", de:
"Einstellungen der finalen Komposition"};
    o.strName = {en: "Name", de: "Name"};
    o.strWidth = {en: "Width", de: "Breite"};
    o.strHeight = {en: "Height", de: "Höhe"};
    o.strPixelAspect = {en: "Pixel Aspect", de: "Pixel-Seitenverhältnis"};
    o.strDuration = {en: "Duration", de: "Dauer"};
    o.strFramerate = {en: "Framerate", de: "Framerate"};
    o.strPx = {en: "px"};
    o.strSeconds = {en: "s"};
    o.strFPS = {en: "fps"};
    o.strSettings = {en:"Settings", de: "Einstellungen"};
    o.strAddTimestamp = {en: "Add timestamp at the end of comp-name", de: "Zeitstempel an
Kompositionsname anhängen"};
    o.strRandomStart = {en: "Start items at random points in time", de: "Die einzelnen
Elemente an zufälligen Zeitpunkten beginnen lassen"};
    o.strLoopItems = {en: String("..and loop each item until its original
duration").addSlashes(), de: "..und diese bis zur ihrer ursprünglichen Länge
wiederholen"};
    o.strOpenComp = {en: "Open composition after successful creation", de: "Öffne
Komposition nach erfolgreicher Erstellung"};
    o.strGenerateComp = {en: "Generate Compositon", de: "Generiere Komposition"};
    o.strFillInAllFields = {en: "Please fill in all input fields.", de: "Bitte füllen Sie
alle Textfelder aus."};
    o.strHelpText =
    {
        en: "Copyright (c) 2013 Sebastian Mayrhuber.\n" +
        "All rights reserved.\n" +
        "\n" +
        "This script allows you to create a randomly composed video from elements that
you provide. Those assets can be anything that can be placed on the After Effects
timeline, including compositions, movie files, image files, audio files etc.\n" +
        "\n" +
        "Guide:\n" +
        "To make the script work, the assets have to be placed inside folders which
follow this simple naming convention: 'layer1', 'layer2' and so on. The reason behind
this is that the script is able to stack assets on top of each other. Assets placed
inside 'layer2' will appear above assets from 'layer1', assets from 'layer3' above assets
from 'layer2' and so on. The number of folders is not limited. The elements themselves
can be named arbitrarily. The upper part of the user interface allows you to configure
the composition to be created whereas in the lower part the script's behavior can be
adjusted.\n" +
        "\n" +
        "Note 1: This version of the script requires After Effects CS5 or later. It can
be used as a dockable panel by placing the script in a ScriptUI Panels subfolder of the
Scripts folder, and then choosing this script from the Window menu.\n" +
        "\n" +
        "Note 2: The setting 'Open comp after successful creation' uses an undocumented
function of After Effects. It should not cause any problems but if you run into any
errors just disable it."
    };

    // Just for testing: Manually setting the current local language
//~     $.locale = "de";  // Restore by setting var to null!


    //
    // FUNCTIONS
    //

    /**
     * buildUI()
     *
     * Description:
     * This function builds the user interface.
     *
     * Parameters:
     * thisObj - Panel object (if script is launched from Window menu); null otherwise.
     *
     * Returns:
     * Window or Panel object representing the built user interface.
     */
      function buildUI(thisObj)
      {
            var pal = (thisObj instanceof Panel) ? thisObj : new Window("palette",
o.scriptName, undefined, {resizeable:true});

            if (pal !== null)
        {
            // Dev-Info: The following string is enclosed in 3 quotes > this way
multiline strings don't need a backslah ("\") at the end of each line!
            var res =
            """Group
                orientation:'column', alignment:['fill', 'fill'], alignChildren:['left',
'top']
                header: Group
                    orientation:'row',  alignment:['fill', 'top'], margins:[0, 0, 0, 10]
                    title: StaticText { text:'""" + o.scriptTitle + """' }
                    help: Button { text:'""" + localize(o.strHelp) + """', maximumSize:
[30, 20], alignment:['right', 'center'] }
                }
                mainCompPnl: Panel
                    text:'""" + localize(o.strConfigureFinalComposition) + """',
orientation:'row', alignment:['fill', 'fill'], alignChildren:['left', 'top'], margins:20
                    lblGrp: Group
                        orientation:'column', alignChildren:['right', 'top']
                        g1: Group { margins:[0, 2, 0, 0]
                            lblName: StaticText { text:'""" + localize(o.strName) +
""":' }
                        }
                        g2: Group { margins:[0, 5, 0, 0]
                            lblWidth: StaticText { text:'""" + localize(o.strWidth) +
""":' }
                        }
                        g3: Group { margins:[0, 5, 0, 0]
                            lblHeight: StaticText { text:'""" + localize(o.strHeight) +
""":' }
                        }
                        g4: Group { margins:[0, 5, 0, 0]
                            lblPixelAspect: StaticText { text:'""" +
localize(o.strPixelAspect) + """:' }
```

```
                        }
                        g5: Group { margins:[0, 5, 0, 0]
                            lblDuration: StaticText { text:'""" + localize(o.strDuration)
+ """:' }
                        }
                        g6: Group { margins:[0, 5, 0, 0]
                            lblFramerate: StaticText { text:'""" +
localize(o.strFramerate) + """:' }
                        }
                    }
                    txtGrp: Group
                        orientation:'column', alignment:['fill','fill'], alignChildren:
['left', 'top']
                        txtName: EditText { text:'""" + o.mainCompName + """', alignment:
['fill','top'], characters:20 }
                        g1: Group { orientation:'row'
                            txtWidth: EditText { text:'""" + o.mainCompWidth + """',
characters:5 }

                            g1_1: Group { margins:[-9, 0, 0, 0]
                                lblPixel: StaticText { text:'""" + localize(o.strPx) +
"""' }

                        }
                        g2: Group { orientation:'row'
                            txtHeight: EditText { text:'""" + o.mainCompHeight + """',
characters:5 }

                            g2_2: Group { margins:[-9, 0, 0, 0]
                                lblPixel: StaticText { text:'""" + localize(o.strPx) +
"""' }

                        }
                        txtPixelAspect: EditText { text:'""" + o.mainCompPixelAspect +
"""', characters:5 }
                        g3: Group { orientation:'row'
                            txtDuration: EditText { text:'""" + o.mainCompDuration +
"""', characters:5 }
                            g3_3: Group { margins:[-9, 0, 0, 0]
                                lblSeconds: StaticText { text:'""" +
localize(o.strSeconds) + """' }

                        }
                        g4: Group { orientation:'row'
                            txtFramerate: EditText { text:'""" + o.mainCompFramerate +
"""', characters:5 }
                            g4_4: Group { margins:[-9, 0, 0, 0]
                                lblFPS: StaticText { text:'""" + localize(o.strFPS) +
"""' }

                        }
                    }
                }

                settingsPnl: Panel
                    text:'""" + localize(o.strSettings) + """', orientation:'column',
alignment:['fill', 'fill'], alignChildren:['left', 'top'], margins:20
                    grpTimestamp: Group
                        orientation:'row', alignment:['left', 'top'], alignChildren:
['left', 'top']
                        cbTimestamp: Checkbox { value:""" + o.cbTimestamp + """ }
                        lbl: StaticText { text:'""" + localize(o.strAddTimestamp) +
"""' }
                    }
                    grpRandomStart: Group
                        orientation:'row', alignment:['left', 'top'], alignChildren:
['left', 'top']
                        cbRandomStart: Checkbox { value:""" + o.cbRandomStart + """ }
```

```
                            lbl: StaticText { text:'""" + localize(o.strRandomStart) +
"""' }
                        }
                        grpLoopItems: Group
                            orientation:'row', alignment:['left', 'top'], alignChildren:
['left', 'top'], margins:[15, 0, 0, 0]
                            cbLoopItems: Checkbox { value:""" + o.cbLoopItems + """ }
                            lbl: StaticText { text:'""" + localize(o.strLoopItems) + """' }

                        grpOpenComp: Group
                            orientation:'row', alignment:['left', 'top'], alignChildren:
['left', 'top']
                            cbOpenComp: Checkbox { value:""" + o.cbOpenComp + """ }
                            lbl: StaticText { text:'""" + localize(o.strOpenComp) + """' }


                    grpGenerateComp: Group
                        orientation:'row', alignment:['fill', 'top'], alignChildren:['right',
'bottom']
                        generateCompBtn: Button { text:'""" + localize(o.strGenerateComp) +
"""', minimumSize:[150, 20] }

            }""";

            // Add the previously created 'resource specification'-string to the main
panel and SAVE returned object for later access
            pal.mainGrp = pal.add(res);

            // Re-'Layout()' and 'Resize()' and also do this on resizing the panel
            pal.layout.layout(true);
            pal.mainGrp.minimumSize = pal.mainGrp.size;
            pal.layout.resize();
            pal.onResizing = pal.onResize = function () { this.layout.resize(); }

            // EVENT-HANDLER
            pal.mainGrp.header.help.onClick = function () { alert(o.scriptTitle + "\n" +
localize(o.strHelpText), o.scriptName); };
            pal.mainGrp.settingsPnl.grpRandomStart.cbRandomStart.onClick =
onClickCbRandomStart;
            pal.mainGrp.grpGenerateComp.generateCompBtn.onClick = generateComposition;

            return pal;
        }
    }

    /**
     * onClickCbRandomStart()
     *
     * Description:
     *
     * Parameters:
     * None.
     *
     * Returns:
     * Nothing.
     */
    function onClickCbRandomStart()
    {
        updateCheckboxStates();
    }

    /**
     * updateCheckboxStates()
     *
     * Description:
     *
     * Parameters:
     * None.
     *
     * Returns:
```

```
     * Nothing.
     */
    function updateCheckboxStates()
    {
        var cbRandomStart = o.sm_gacPal.mainGrp.settingsPnl.grpRandomStart.cbRandomStart;
// ..just for easier referencing in this function
        var cbLoopItems = o.sm_gacPal.mainGrp.settingsPnl.grpLoopItems.cbLoopItems;
// ..just for easier referencing in this function

        if (cbRandomStart.value)
            cbLoopItems.enabled = true;
        else
        {
            cbLoopItems.value = false;
            cbLoopItems.enabled = false;
        }
    }

    /**
     * generateComposition()
     *
     * Description:
     *
     * Parameters:
     * None.
     *
     * Returns:
     * Nothing.
     */
    function generateComposition()
    {
        // CHECK user input for correctness and completeness
        var allOK = true;
        var timestamp = "";
        var randomStart = false;
        var txtGrp = o.sm_gacPal.mainGrp.mainCompPnl.txtGrp;  // ..just for easier
referencing in this function
        var settingsPnl = o.sm_gacPal.mainGrp.settingsPnl;  // ..just for easier
referencing in this function

        if (txtGrp.txtName.text == "" ||
            txtGrp.g1.txtWidth.text == "" ||
            txtGrp.g2.txtHeight.text == "" ||
            txtGrp.txtPixelAspect.text == "" ||
            txtGrp.g3.txtDuration.text == "" ||
            txtGrp.g4.txtFramerate.text == "")
        {
            allOK = false;
            alert(localize(o.strFillInAllFields), o.scriptName);
        }

        if (!allOK)
            return;

        // Check setting: Add timestamp to comp-name?
        if (settingsPnl.grpTimestamp.cbTimestamp.value)
        {
            var now = new Date();
            timestamp = "_";
            timestamp += (now.getHours() < 10) ? "0" : "";  timestamp += now.getHours();
            timestamp += (now.getMinutes() < 10) ? "0" : "";    timestamp +=
now.getMinutes();
            timestamp += (now.getSeconds() < 10) ? "0" : "";    timestamp +=
now.getSeconds();
//~             timestamp = "_" + Math.round(new Date().getTime() / 1000);  // ..or Unix-
Timestamp (in seconds)
        }

        // Treat as a single undo-able event
        app.beginUndoGroup("GenArtsComposer");

        // Search through project and store references to all folders named 'layer1',
'layer2', and so on in an array
        o.clipFolders = new Array();
        for (var i=1; i<=app.project.numItems; i++)
        {
            if ( (app.project.item(i) instanceof FolderItem) && (new
RegExp(/^layer\d+/).test(app.project.item(i).name)))
                o.clipFolders.push(app.project.item(i));

            // Deselect every existing 'main composition' > newly generated main-comp
gets selected later on for easier spotting inside the panel
            if (app.project.item(i).name.indexOf( txtGrp.txtName.text ) != -1)
                app.project.item(i).selected = false;
        }

        // Just for debugging: Log found clipFolders into JS-console
        for (i=0; i<o.clipFolders.length; i++)
            $.writeln("o.clipFolders[" + i + "] = " + o.clipFolders[i].toString() + " ,
name = " + o.clipFolders[i].name);

        // Create 'main' composition that holds all the clips and 'select' it afterwards
(for easier spotting in the project panel)
        o.mainComp = app.project.items.addComp(txtGrp.txtName.text + timestamp,
Number(txtGrp.g1.txtWidth.text), Number(txtGrp.g2.txtHeight.text),
Number(txtGrp.txtPixelAspect.text), Number(txtGrp.g3.txtDuration.text),
Number(txtGrp.g4.txtFramerate.text));
        o.mainComp.selected = true;

        // Loop through all 'layer'-folders and add randomly selected items from every
folder to the main-comp.
        // > Random items will be added one after another until the timeline is
completely filled  with 'material'.
        for (var i=0; i<o.clipFolders.length; i++)
        {
            var curLength = 0;  // helper-var used for 'filling' the timeline

            if (o.clipFolders[i].numItems > 0)
            {
                while (curLength < o.mainComp.duration)
                {
                    // Get random item from layer-folder
                    var randomItem = o.clipFolders[i].item( getRandomInt(1,
o.clipFolders[i].numItems) );
                    var itemLayer = o.mainComp.layers.add(randomItem);

                    // Random item is a COMPOSITION
                    if (randomItem instanceof CompItem)
                    {
                        // Check setting: Start items at random points in time?
                        if (settingsPnl.grpRandomStart.cbRandomStart.value)
                        {
                            o.cbRandomStart = true;

                            // Check setting: When starting items at random points in
time: loop each item until it's original duration?
                            if (settingsPnl.grpLoopItems.cbLoopItems.value)
                            {
                                o.cbLoopItems = true;

                                // Make sure the items are placed one after another in
the timeline using a helper-variable 'curLength'
                                itemLayer.startTime = curLength;

                                // Activate Timeremapping
                                itemLayer.timeRemapEnabled = true;

                                // Add an expression to the timeRemap property that lets
the layer start at a random position in time and loop if it ends
```

```
                                var expString = "seedRandom(index, true);\n" +
                                "d = source.duration;\n" +
                                "offset = random(d);\n" +
                                "(time + offset)%d";
                                itemLayer.property("ADBE Time Remapping").expression =
expString;

                                // Update helper-variable 'curLength'
                                curLength += randomItem.duration;
                            }
                            else
                            {
                                 o.cbLoopItems = false;

                                 // Set random inPoint
                                 itemLayer.inPoint = getRandomArbitary(0,
randomItem.duration);

                                // Make sure the items are placed one after another in
the timeline using a helper-variable 'curLength'
                                itemLayer.startTime = curLength - itemLayer.inPoint;

                                // Update helper-variable 'curLength'
                                curLength = itemLayer.outPoint;
                            }

                        } // END if (settingsPnl.grpRandomStart.cbRandomStart.value)
                        else
                        {
                            o.cbRandomStart = false;

                            // Make sure the items are placed one after another in the
timeline using a helper-variable 'curLength'
                            itemLayer.startTime = curLength;

                            // Update helper-variable 'curLength'
                            curLength += randomItem.duration;
                        }

                    } // END if (randomItem instanceof CompItem)

                     // Random item is a FOOTAGE-ITEM
                    if (randomItem instanceof FootageItem)
                    {
                        // Make sure the items are placed one after another in the
timeline using a helper-variable 'curLength'
                        itemLayer.inPoint = curLength;

                        // Set random footage-item-duration
                        itemLayer.outPoint = itemLayer.inPoint + getRandomArbitary(1, 5);

                        // Update helper-variable 'curLength'
                        curLength = itemLayer.outPoint;
                    }

                } // END while (curLength < o.mainComp.duration)
            } // END if (o.clipFolders[i].numItems > 0)
        } // END for (var i=0; i<o.clipFolders.length; i++)


        // Check setting: Open composition after successful creation?
        if (settingsPnl.grpOpenComp.cbOpenComp.value)
            openCompPanel(o.mainComp);

        // End of the actions that make the single undoable event
        app.endUndoGroup();
    }


    //
    // MAIN CODE
    //
    // AE-Version check
      if (parseFloat(app.version) < 10.0)
           alert(localize(o.strMinAE100), o.scriptTitle) ;
      else
    {
        // Build and show the palette
        o.sm_gacPal = buildUI(thisObj);
        if (o.sm_gacPal !== null)
        {
            // Update UI values, if saved in the settings
            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_compName"))
                o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.txtName.text =
app.settings.getSetting("sm", "sm_GenArtsComposer_compName");

            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_compWidth"))
                o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.g1.txtWidth.text =
app.settings.getSetting("sm", "sm_GenArtsComposer_compWidth");

            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_compHeight"))
                o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.g2.txtHeight.text =
app.settings.getSetting("sm", "sm_GenArtsComposer_compHeight");

            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_compPixelAspect"))
                o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.txtPixelAspect.text =
app.settings.getSetting("sm", "sm_GenArtsComposer_compPixelAspect");

            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_compDuration"))
               o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.g3.txtDuration.text =
app.settings.getSetting("sm", "sm_GenArtsComposer_compDuration");

            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_compFramerate"))
                o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.g4.txtFramerate.text =
app.settings.getSetting("sm", "sm_GenArtsComposer_compFramerate");

            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_timestamp"))
                o.sm_gacPal.mainGrp.settingsPnl.grpTimestamp.cbTimestamp.value =
(app.settings.getSetting("sm", "sm_GenArtsComposer_timestamp") == "false") ? false :
true;

            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_randomStart"))
                o.sm_gacPal.mainGrp.settingsPnl.grpRandomStart.cbRandomStart.value =
(app.settings.getSetting("sm", "sm_GenArtsComposer_randomStart") == "false") ? false :
true;

            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_loopItems"))
                o.sm_gacPal.mainGrp.settingsPnl.grpLoopItems.cbLoopItems.value =
(app.settings.getSetting("sm", "sm_GenArtsComposer_loopItems") == "false") ? false :
true;

            updateCheckboxStates();

            if (app.settings.haveSetting("sm", "sm_GenArtsComposer_openComp"))
            o.sm_gacPal.mainGrp.settingsPnl.grpOpenComp.cbOpenComp.value =
(app.settings.getSetting("sm", "sm_GenArtsComposer_openComp") == "false") ? false : true;

            // Save current UI settings upon closing the palette
            o.sm_gacPal.onClose = function()
            {
                app.settings.saveSetting("sm", "sm_GenArtsComposer_compName",
o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.txtName.text);
                app.settings.saveSetting("sm", "sm_GenArtsComposer_compWidth",
o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.g1.txtWidth.text);
                app.settings.saveSetting("sm", "sm_GenArtsComposer_compHeight",
o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.g2.txtHeight.text);
                app.settings.saveSetting("sm", "sm_GenArtsComposer_compPixelAspect",
o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.txtPixelAspect.text);
                app.settings.saveSetting("sm", "sm_GenArtsComposer_compDuration",
o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.g3.txtDuration.text);
                app.settings.saveSetting("sm", "sm_GenArtsComposer_compFramerate",
o.sm_gacPal.mainGrp.mainCompPnl.txtGrp.g4.txtFramerate.text);

                app.settings.saveSetting("sm", "sm_GenArtsComposer_timestamp",
o.sm_gacPal.mainGrp.settingsPnl.grpTimestamp.cbTimestamp.value);
                app.settings.saveSetting("sm", "sm_GenArtsComposer_randomStart",
o.sm_gacPal.mainGrp.settingsPnl.grpRandomStart.cbRandomStart.value);
                app.settings.saveSetting("sm", "sm_GenArtsComposer_loopItems",
o.sm_gacPal.mainGrp.settingsPnl.grpLoopItems.cbLoopItems.value);
                app.settings.saveSetting("sm", "sm_GenArtsComposer_openComp",
o.sm_gacPal.mainGrp.settingsPnl.grpOpenComp.cbOpenComp.value);
            }

            if (o.sm_gacPal instanceof Window)
            {
                // Show the palette
                o.sm_gacPal.center();
                o.sm_gacPal.show();
            }
            else
                o.sm_gacPal.layout.layout(true);

        }  // if (o.sm_gacPal !== null)
    }  // else AE-Version check

})(this);
```