

INHALTSÜBERSICHT

Einleitung	2
Worum geht es hier?	2
Skriptsprache warum?	2
Welche Programmiersprache?	3
Hilfe!	3
Entwicklungsumgebung	4
Dokumentation	5
Wie kommt der Skript-Quelltext und Photoshop CS zusammen?	5
Installation	5
Skript-Aufruf in Photoshop	5
JavaScript	5
Speicherorte	5
Dateien	6
Elementare Grundstrukturen von JavaScript	6
Syntax	6
Werte	7
Objekte	7
Eigenschaften	7
Methoden	7
Kommentare	8
Variablen	8
Datenfelder	8
Datentypen	9
Arithmetische Operatoren von Zahlenoperanden	9
Boolsche Operanden	9
Logische Operatoren	9
Zuweisungsoperatoren	9
Auswahlstrukturen	10
Die if-else Bedingung	10
Die switch-Fallunterscheidung	10
Schleifenstrukturen	11
Die while-Schleife	11
Die do-while-Schleife	11
Die for-Schleife	11
Die for-in-Schleife	11

Gezielte Abbrüche von Schleifen mit break und continue.....	11
Funktionen	12
try-catch Fehleranweisung	12
with-Anweisung	12
Erstes Anwendungsbeispiel	13
Aufgabenstellung.....	13
Programmbeschreibung	13
Dialogfenster	13
Quelltext mit Beschreibung.....	13
Anhang	16
Download	16
Lizenz.....	16

Einleitung

Worum geht es hier?

Dieses JavaScript-Grundlagen-Tutorial ist für Photoshop-CS-Nutzer gedacht, die die Automatisierungstechnik mittels JavaScript auf eine produktivere Stufe heben möchten. Programmierkenntnisse werden nicht vorausgesetzt, jedoch Interesse und Lernbereitschaft sind selbstverständlich förderlich. Zunächst erfahren Sie, welche Möglichkeiten es im Zusammenspiel mit Photoshop und Programmiersprachen gibt, danach wird ein Basiskurs in JavaScript vermittelt, s. d. Sie in die Lage versetzt werden eine eigene JavaScript-Datei zu erstellen und in Photoshop CS anzuwenden. Den Abschluss bildet ein konkretes einfaches Anwendungsbeispiel in JavaScript, das Schritt für Schritt erklärt wird. Als Zugabe wird der komplette Quellcode dieses Beispiels als Downloaddatei zur Verfügung gestellt.

Skriptsprache warum?

Es gibt doch **Aktionen**, um immer wiederkehrende Aufgaben zu erledigen. Hierbei werden die Abfolgen mit allen eingegebenen Werten einfach aufgezeichnet und können per Mausklick immer wieder einheitlich als Stapelverarbeitung abgespult werden. Warum sollte man sich also dann überhaupt mit einer Skriptsprache beschäftigen?

Weil JavaScript zusätzlich die Möglichkeit bietet, Bilddokumente auszuwerten und in Abhängigkeit von deren Eigenschaften, wie z. B. Hochformat oder Querformat, Entscheidungen zu treffen und ganz spezifisch bei den auszuführenden Bildbearbeitungsschritten darauf zu reagieren. In JavaScript hat man die Möglichkeit Entscheidungen zu treffen und das ist die wesentliche Stärke von Skripts.

Nachteil von JavaScript, es muss halt programmiert werden und ist somit ein höherer Arbeitsaufwand und die Einarbeitungszeit sollte auch nicht verschwiegen werden. Manchen Nutzern - wie mir - macht es jedoch auch Vergnügen, sich mit einer Programmiersprache zu beschäftigen.

Welche Programmiersprache?

Bevor man loslegt, sollte man eine Frage klären: Womit programmiert man überhaupt?

Es stehen drei Möglichkeiten - Abläufe in Photoshop per Skript zu automatisieren - zur Verfügung:

- **VBScript** (nur auf Windows)
- **AppleScript** (nur auf Mac)
- **JavaScript** (sowohl auf Windows, als auch auf Mac)

Wir werden uns in diesem Tutorial, wie die Überschrift schon "befürchten lässt", ausschließlich mit **JavaScript**, basierend auf der Windows-Plattform beschäftigen. Bei dieser Entscheidung ist mir kein Nachteil gegenüber den beiden übrigen Programmiersprachen bekannt, da Adobe selbst offensichtlich auch auf das Potenzial von JavaScript setzt. Ferner hat JavaScript durch seinen vielfältigen Einsatz im Web eine sehr weite Verbreitung gefunden und besitzt eine sehr übersichtliche Struktur. JavaScript besteht aus Objekten die Eigenschaften und Methoden besitzen. Das war eigentlich schon alles.

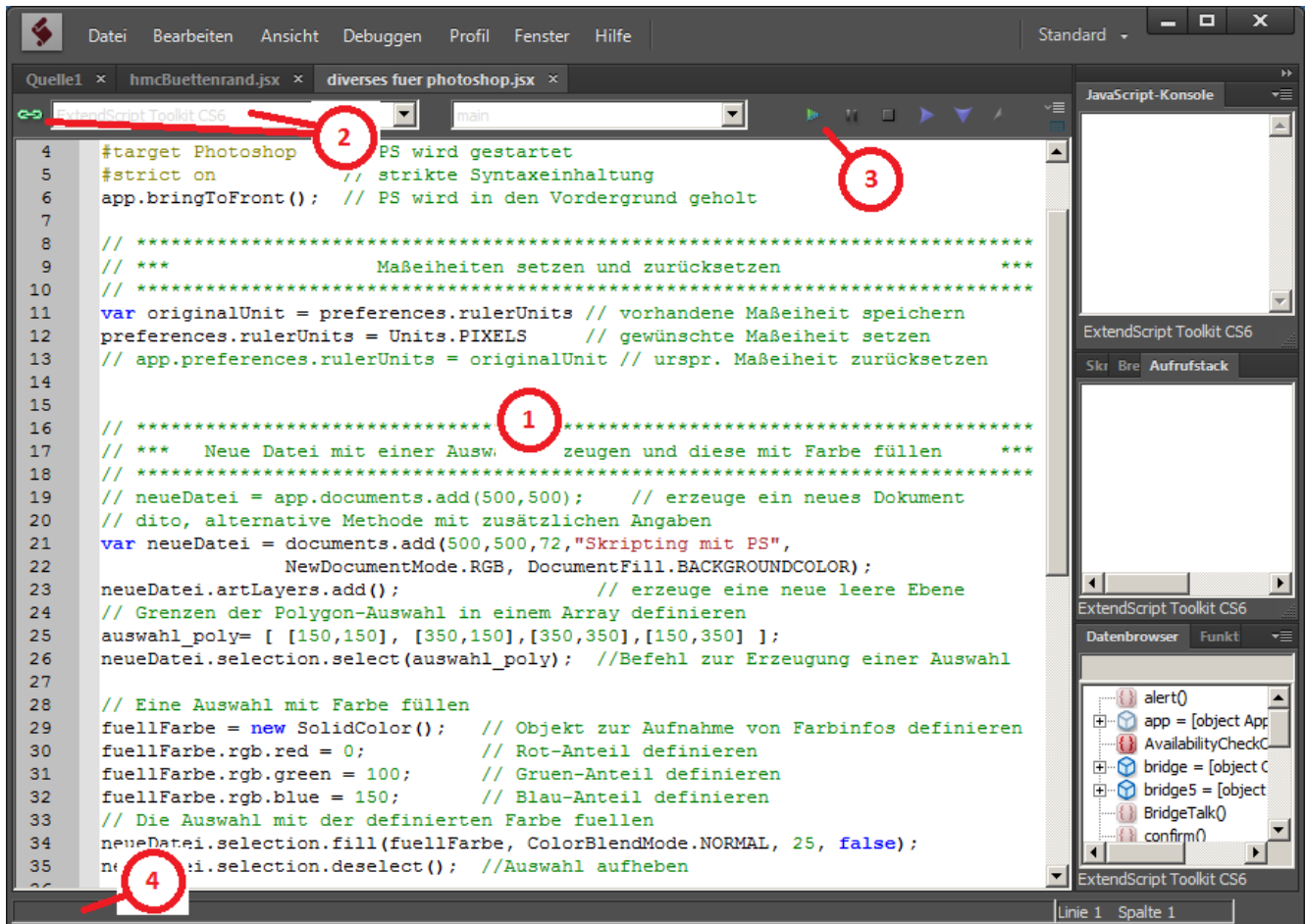
Hilfe!

JavaScript ist eine Skriptsprache, die im Grunde keine Entwicklungsumgebung benötigt, d. h. es ist kein Hilfebutton, Debugger, etc. - wie bei den integrierten Entwicklungsumgebungen (IDE's) der "richtigen" Programmiersprachen - vorhanden. Bei der Erstellung und Testung der Skripte mit einem normalen Texteditor ist man jedoch relativ hilflos, vorhandene Fehler aufzuspüren.

Bei der Erstellung und Testung wird man allerdings unterstützt, wenn man das **Adobe® Extendscript Toolkit CS** verwendet, es wird mit Photoshop CS mit installiert. Hiermit kann man Skripte komfortabel entwickeln, sofort an Photoshop zum Start übergeben und sich auch bei der Fehlersuche durch ein Debugger unterstützen lassen. Eine umfangreiche Hilfe ist auch enthalten. In den Palettenfenstern werden die in Photoshop verfügbaren Anweisungen, Fehlern etc. eingeblendet. Im Hauptfenster findet die eigentliche Scripting-Arbeit statt, mit Syntaxhighlighting und Vervollständigung von Befehlen. Es ist sozusagen die "Photoshop-JavaScript-Entwicklungsumgebung".

Entwicklungsumgebung

Gestartet wird diese im Windows-Betriebssystem über: "Start > Programme > **Adobe ExtendScript Toolkit CS6**"



Legende:

1. Hier kommt der Programmcode rein
2. Hier wird die Ziellanwendung ausgewählt (Extendscript Toolkit, Bridge oder Photoshop), links daneben wird der Verbindungsstatus über das kleine Symbol angezeigt, durch Anklicken wird die Verbindung zur Ziellanwendung hergestellt.
Anmerkung: Programmfehler entstehen häufig dadurch, dass dort nicht die korrekte Ziellanwendung ausgewählt wurde. Um diese Fehlerquelle zu umgehen, können Sie auch im Skript-Quellcode die Anweisung **#target Photoshop** setzen, damit wird Photoshop beim Programmstart automatisch in den Vordergrund geholt.
3. Skriptausführung starten.
4. Statuszeile, hier werden z. B. bei Betätigung der Funktionstaste **[F7]** eventuelle Syntaxfehler angezeigt.
 - Die Fenster auf der rechten Seite können individuell angeordnet werden.
 - Eine Suchfunktion **[Strg+F]** hilft beim Suchen/Ersetzen, auch Rückgängig **[Strg+Z]** und Wiederherstellen erleichtert die Tipparbeit deutlich.
 - Zeilennummern und Syntaxhighlighting um Java-Schlüsselwörter farbig hervorzuheben etc. sind selbstverständlich enthalten.

- Um das Skript zu prüfen (debuggen) sind alle notwendigen Funktionen, wie Einzelschritt, Stoppen, Breakpoints, etc., sogar Variablenwerte werden angezeigt, wie bei den "großen" IDE's. Es ist eigentlich alles da, was benötigt wird.

Dokumentation

Bei der Installation von Photoshop werden die beiden nachstehenden Dokumentationen mit installiert. Der Aufruf erfolgt im Extendscript Toolkit über Menü > Hilfe...

- Menü > Hilfe > **Adobe Skripte-Einführung** (ADOBE® INTRODUCTION TO SCRIPTING)
- Menü > Hilfe > **JavaScript Tools Guide CS6**

Weitere Dokumentation:

- Das Praxisbuch von Petra Kriesinger, "**Photoshop mit JavaScript steuern**" ISBN 3-7723-7307-0. Dieses Buch ist von 2005 und leider nur noch gebraucht zu bekommen, es bietet einen guten Einstieg sowie anschauliche Erklärungen und Beispiele, die sich auf die älteren Versionen PS7 und CS2 beziehen.
- Das Online-Angebot von **SELFHTML**, das Sie ebenfalls kennen sollten!
Link: de.selfhtml.org/javascript/index.htm
Dieser Abschnitt dokumentiert die Programmiersprache JavaScript
- Ein Buch über den Umgang mit JavaScript, es muss nicht die neuste Auflage sein, ist natürlich auch förderlich.

Wie kommt der Skript-Quelltext und Photoshop CS zusammen?

Installation

Die Installation ist sehr simpel, die fertige '*Skriptdatei.jsx*' wird in das Verzeichnis 'c:\Programme\Adobe\Adobe Photoshop CS6\Presets**Scripts**' kopiert. Das war schon alles!

Skript-Aufruf in Photoshop

Nachdem das Skript in das zuvor beschriebene Verzeichnis kopiert wurde, kann es nach einem Neustart von Photoshop CS aufrufen und angewendet werden. Der Aufruf erfolgt in Photoshop über den -Menübefehl: 'Menü > Datei > **Skripten**'. Jetzt kann das gewünschte Skript, über den Dateinamen ausgewählt und aufgerufen werden. Sollte der Dateiname nicht sichtbar sein, so besteht auch die Möglichkeit, das Skript auch über den Menübefehl '**Durchsuchen...**', zu laden.

JavaScript

Speicherorte

Skripte können grundsätzlich in jedem beliebigen Verzeichnis gespeichert werden!

Wenn Skripte direkt aus Photoshop gestartet werden sollen, ist es ratsam, diese in das dafür vorgesehene Verzeichnis: "C:\Programme\Adobe\Adobe Photoshop CS6\Presets**Scripts**" abzulegen, nach einem Neustart von PS sind alle Skript-Dateien die dort liegen, sichtbar und können ausgewählt bzw. gestartet werden.

JavaScript-Grundlagen-Tutorial für Photoshop CS

In der Entwicklungsphase hingegen sollten Sie ein separates Arbeitsverzeichnis anlegen. Wegen der besseren Übersicht, sollten Sie für jedes Projekt sogar einen eigenen Ordner anlegen. Die Ausführung von Skripten erfolgt aus der Entwicklungsumgebung **ExtendScript Toolkit CS6**, über die Funktionstaste **[F5]** oder über das Menü > Debuggen > **Ausführen**. Ist Photoshop als Zielanwendung ausgewählt so wird Photoshop in Vordergrund geholt. Nach Abarbeitung der Skriptanweisungen erhält die Entwicklungsumgebung automatisch wieder den Fokus.

Dateien

- JavaScript-Dateien für Photoshop werden mit der Dateiendung **.jsx** gespeichert! Es sind reine Textdateien und können mit jedem beliebigen Texteditor gelesen bzw. editiert werden.
- Zusätzlich gibt es noch eine weitere Dateiendung **.jsxinc** für Skripte. Hierbei handelt es sich um externe/ausgelagerte Skriptdateien, die mit der **#include**-Anweisung eingebunden und beim Programmstart geladen werden. Bei umfangreicheren Scriptings macht es Sinn mit ausgelagerten Dateien, wegen der besseren Quellcode-Übersicht und wegen der Mehrfachverwendung, zu arbeiten. In dem nachstehenden Beispiel wird die rechte ausgelagerte Datei, in die linke, durch die **#include**-Anweisung eingebunden.

Hauptskript. jsx #include MeineAuslagerungsdatei. jsxinc Anweisung2	MeineAuslagerungsdatei. jsxinc Anweisung1 Anweisung2
--	---

Elementare Grundstrukturen von JavaScript

Syntax

Der Syntax beschreibt den genauen Aufbau, die Grammatik und auch die Rechtschreibung dieser Sprache. Grundsätzlich ist folgendes zu beachten:

- Die Sprache ist Case-Sensitive, das bedeutet, dass Namen immer gleich geschrieben werden müssen. Beispielsweise besteht ein Unterschied zwischen der Variablen `meineVariable` und der Variablen `MeineVariable`.
- Selbstvergebene Namen dürfen keine Leer- oder Sonderzeichen beinhalten.
- Selbstvergebene Namen sollten mit einem Buchstaben beginnen und kein ä, ö, ü oder ß enthalten.
- Selbstvergebene Namen dürfen nicht mit einem reservierten Wort übereinstimmen.
- Namen dürfen nicht doppelt vergeben werden, d.h. wenn z.B. eine Funktion `xyz` heißt, darf es keine Variable `xyz` geben und umgekehrt.

Grundsätzlich könnte man sagen, dass JavaScript zeilenweise aufgebaut ist. Jede Anweisung sollte innerhalb einer Zeile platziert werden und mit einem Semikolon (;) beendet werden. Das Semikolon ist jedoch nicht Pflicht. Unbedingt notwendig wird es erst, wenn mehrere Anweisungen innerhalb einer Zeile stehen sollen - hier muss zur Abgrenzung der einzelnen Anweisungen das Semikolon als Trennzeichen verwendet werden.

Beispiel:

```
// ohne Semikolon werden die Anweisungen durch das
// Zeilenende voneinander getrennt ...
Anweisung1
Anweisung2
// das Semikolon wäre hier jedoch auch nicht falsch ...
Anweisung3 ;
Anweisung4 ;
```

```
// hier muss das Zeichen unbedingt gesetzt werden ...  
Anweisung5 ; Anweisung6 ; ; Anweisung7
```

Werte

Bei der Verwendung von Werten gibt es bestimmte Regeln, an die es sich zu halten gilt. Diese sind wie folgt:

- Text (String) muss immer mit Anführungszeichen (" ... " oder ' ... ') umschlossen sein. Dabei muss das Anfangs-Zeichen gleich dem Endzeichen sein (Also "text" oder 'text' aber nicht 'text" oder "text'). Sollen innerhalb des Textes die gleichen Anführungszeichen verwendet werden, so müssen diese als Sonderzeichen deklariert sein (aus " wird \" und aus ' wird \').
- Zahlen werden ohne Anführungszeichen notiert (z. B. 123 oder 567). Kommazahlen werden mit Punkt statt mit Komma notiert (also 1.23 statt 5,67).
- Boolesche Werte (Wahrheitswerte) werden mit **true** für wahr und **false** für falsch notiert, ohne Anführungszeichen.
- **Typen, Objekte** und **Variablen** werden lediglich mit Namen und ohne Anführungszeichen notiert (z.B. String oder window).

Objekte

Es ist ein allgemeiner Datentyp. Objekte sind fest definierte Dinge, dazu gehören u. a. Bibliotheken, sie enthalten wichtige Eigenschaften und Methoden die zur Verarbeitung notwendig sind. Objekte enthalten Eigenschaften und Methoden die die Verarbeitung des jeweiligen Objektes oder eines anderen Objektes ermöglichen. Ein Objekt kann unter Umständen auch aus einem Objekt und seinen (mehreren) Unterobjekten bestehen. Diese werden durch einen Punkt (.) vom Objekt getrennt.

Eigenschaften

Eine Eigenschaft gibt einen bestimmten Wert eines Objektes wieder. Um diesen abzufragen notiert man das Objekt, einen Punkt (.) und die Eigenschaft.

Beispiele: `tischplatte.hight` oder `schrack.tuer.color`

Erklärung:

Das Objekt `tischplatte`. Die Eigenschaft die abgefragt wird heißt `hight`. Als Resultat könnte man also die Höhe des Tisches erhalten (alle anderen Eigenschaften, beispielsweise die Breite des Tisches wären dabei vernachlässigt). Nächstes Beispiel heißt das Objekt `schrack`. Dazu wird das Unterobjekt `tuer` benannt und von diesem die Eigenschaft `color` abgefragt. Als Resultat könnte man also von diesem Schrack die Farbe der Tür erhalten (alle anderen Farben, beispielsweise die der Schubkästen, wären dabei vernachlässigt).

Methoden

Eine Methode ist eine Funktion eines Objektes. Eine Methode wird, ebenso wie eine Eigenschaft, durch einen Punkt (.) getrennt an das jeweilige Objekt geschrieben (man sagt notiert). Einer Methode können dazu bestimmte Werte (Parameter) übergeben werden, die die Methode zur Verarbeitung benötigt. Diese Werte werden innerhalb von zwei Klammern ((und)) an die Methode heran notiert. Mehrere Werte werden dabei durch Kommata getrennt. Benötigt die Methode keine weiteren Werte, so sollten diese Klammern dennoch gesetzt werden (nur eben ohne Inhalt).

Beispiel: `tisch.chanceColor()` oder `buch.calcPages(10,30)`

Erklärung:

Wird mit dem Objekt `tisch` gearbeitet. Die Methode heißt `changeColor()`. Als Resultat könnte man hierbei erreichen, dass die Tischfarbe geändert wird. Nächstes Beispiel wird mit dem Objekt `buch` gearbeitet. Als Methode wird `calcPages()` verwendet. Als Resultat könnte man die Anzahl der Zeichen der Seiten 10 bis 30 als Ergebnis zurück bekommen.

Kommentare

Kommentare im Programmcode sind unerlässlich, um das Skript übersichtlich und verständlich zu gestalten, ich kann nur dringlich empfehlen den Quellcode sehr gut zu kommentieren.

Ein einzeiliger Kommentar beginnt mit einem -Doppel-Slash //

Mehrzeilige Kommentare beginnen mit einem Slash, gefolgt von einem Stern `/*` und endet mit den gleichen Zeichen jedoch in umgekehrter Reihenfolge `*/`

Beispiele:

```
// Dies ist ein einzeiliger Kommentar
/* Hier
   steht
   ein
   mehrzeiliger
   Kommentar
*/
```

Variablen

Eine Variable reserviert einen Speicherplatz im Computer, denen zur Laufzeit eines Programms/Skripts temporär Werte zugeordnet werden können. Diese Werte können dabei beliebige Dinge sein: Text, Zahlen, Methoden, Eigenschaften, Objekte, Berechnungen etc.. Auf diesen Speicherplatz kann dann jederzeit über den Namen der Variablen wieder zugegriffen werden und der Inhalt gelesen oder verändert werden. Dabei ist es nicht notwendig eine Variable vorher zu deklarieren (definieren) wie es in anderen Programmiersprachen unerlässlich ist. In JavaScript werden diese Variablen automatisch deklariert. Ebenso ist es auch nicht nötig den Typ einer Variablen (z.B. Integer, String ...) zu definieren, vielmehr kann fließend von einem Typ in den nächsten gesprungen werden, ohne, dass es Fehlermeldungen geben würde.

Angesprochen werden Variablen über den Namen, die sie vorher zugewiesen bekommen haben.

Beispiele: `a = 123; b = "Mein Text";`

Eine Variable ohne Wertzuweisung hat den Wert "undefined", dies kann im Sinne von JavaScript gezielt und sinnvoll verwendet werden.

Namensregeln für Variablen:

- Das erste Zeichen muss ein Buchstabe oder Unterstrich sein
- Es darf kein reserviertes JavaScript-Schlüsselwort verwendet werden

Datenfelder

Ein Array ist eine Sammlung von Variablen, die alle über einen Namen angesprochen werden können.

Beispiel:

```
Tag = new Array(7);
Tag[0] = "Montag", Tag[1] = "Dienstag", Tag[2] = "Mittwoch", Tag[3] =
"Donnerstag", Tag[4] = "Freitag", Tag[5] = "Samstag"; Tag[6] = "Sonntag";
```

Datentypen

JavaScript unterstützt vier Grundtypen:

Boolean	true oder false, (wahr oder falsch)
Number	Dezimal- oder Gleitkommazahlen, bei Gleitkommazahlen muss der Dezimalpunkt verwendet werden
String	Alphanumerische Zeichen
Object	Allg. Datentyp, er wird u. a. für das Speichern von Instanzen und Klassen verwendet. Beispiel: aktuellesDatum = new Date();

Arithmetische Operatoren von Zahlenoperanden

+	Addition	Beispiel: 3 + 4
-	Subtraktion	Beispiel: 4 - 3
-	Negierung	Beispiel: -3
*	Multiplikation	Beispiel: 2 * 3
/	Division	Beispiel: 2 / 3
%	Modulo	Beispiel: 15 % 9, das Ergebnis ist der Rest der Division, also 6
++	Inkrement	Beispiel: zahl ++ erhöht den Wert um 1
--	Dekrement	Beispiel: zahl -- reduziert den Wert um 1

Boolsche Operanden

==	Gleichheit	Beispiel: 3 + 4 = 7 liefert true
!=	Ungleichheit	Beispiel: 8 != 4 + 4 liefert false
<	Kleiner-als	Beispiel: 5 < 6 ergibt true
>	Größer-als	Beispiel: 5 > 6 ergibt false
/	Division	Beispiel: 2 / 3
<=	Kleiner-als oder gleich	Beispiel: 6 <= 6 ergibt true
>=	Größer-als oder gleich	Beispiel: 5 >= 6 ergibt false

Logische Operatoren

&&	AND	Beispiel: (4 + 4 == 8) && (2 + 3 == 5) ergibt true
	OR	Beispiel: (4 + 4 == 8) (2 + 3 == 5) ergibt true
!	NOT	Beispiel: !(4 + 4 == 8) ergibt false
?:	entweder ODER	Beispiel: (3 < 4 ? "ja" : "nein") ergibt "ja"

Zuweisungsoperatoren

+=	Addition	Beispiel: a += 5 alternativ: a = a + 5
-=	Subtraktion	Beispiel: a -= 5 alternativ: a = a - 5
*=	Multiplikation	Beispiel: a *= 5 alternativ: a = a * 5
/=	Division	Beispiel: a /= 5 alternativ: a = a / 5
%=	Modulo	Beispiel: a %= 5 alternativ: a = a % 5
=	Direkte Zuweisung	Beispiel: a = 5

Auswahlstrukturen

Auswahl-/Kontrollstrukturen sind Anwendungen von logischen oder vergleichenden Operatoren und spezielle Anweisungsschritte, mit denen man Entscheidungen über den weiteren Programmablauf vorgeben kann, wenn bestimmte Bedingungen eintreten.

Die if-else Bedingung

Diese Auswahlstruktur gibt es in fast allen Programmiersprachen mit leichten Differenzen der Schreibung.

Beispiele:

Entscheidung zwischen zwei einzelnen Befehlen

```
if (bedingung)
    anweisung1;
else
    anweisung2;
```

Entscheidung zwischen zwei Anweisungsblöcken

```
if (bedingung)
{anweisungen1}
else
{anweisungen2}
```

Der else-Zweig kann entfallen. Für die Formulierung der Bedingung gelten die üblichen Regeln, man sehe die Vergleichsoperatoren. Das Ergebnis muss **true** oder **false** sein.

Die switch-Fallunterscheidung

Diese Auswahlstruktur prüft, ob die hinter `switch` angegebenen Variablenwerte den tatsächlichen entsprechen und führt gegebenenfalls die zugehörige Anweisung bzw. den Anweisungsfolge aus. Der default-Zweig wird dann ausgeführt, wenn es vorher keinen Treffer gab. Wichtig sind die `break`-Anweisungen. Hier schlägt das schwere C-Erbe von JavaScript voll zu. Die Switch-Anweisung darf nicht mit der Select-Case-Anweisung von Basic oder der Case-Of-Anweisung von Pascal verwechselt werden. Denn C und JavaScript freuen sich über einen einmal erzielten Treffer so sehr, dass sie danach alle weiteren Anweisungen der restlichen Case-Zweige ohne Prüfung auch noch ausführen. Ich würde so etwas Irreführung oder einen fetten Programmfehler nennen. Mich fragt aber keiner. Also: erst `break` bringt die `switch`-Anweisung zum switchen.

```
switch (variable)
{
Case "wert1":
    anweisungen1;
    break;
Case "wert2":
    anweisungen2;
    break;
...
default:
    anweisungenx;
}
```

Schleifenstrukturen

Die while-Schleife

Die Schleifenstruktur endet abhängig von einer Bedingung. Der grundlegende Aufbau der Schleife ist: `while (bedingung) befehl;` Oder mehr textlich: Solange die Bedingung erfüllt ist, führe den Befehl aus.

```
while (Bedingung1)
{
    [Anweisungen1;]
    if (Bedingung2) [{Anweisungen2;} break; []]}
    [Anweisungen3;]
    if (Bedingung3) continue;]
    [Anweisungen4;]
}
```

Die do-while-Schleife

Genauso wie die while-Schleife nur anders: Die Bedingung wird erst am Ende geprüft. Eine wichtige Folge ist, diese Schleife wird in jedem Falle mindestens einmal durchlaufen.

```
do
{
    [Anweisungen1;]
    if (Bedingung1) [{Anweisungen2;} break; []]}
    [Anweisungen3;]
    if (Bedingung2) continue;]
    [Anweisungen4;]
}
while (Bedingung3);
```

Die for-Schleife

Wenn die Anzahl erforderlicher Umläufe vorher bekannt ist, hat die For.. -Schleife Vorteile, weil sie schnell und platzsparend abgearbeitet wird. Ihre Syntax stammt aus C und ist für den Basic- oder Pascal-Programmierer (ja, es gibt sogar noch Delphi!) sehr gewöhnungsbedürftig.

```
for ([var] i = von; i <= bis; i++)
{
    [Anweisungen1;]
    if (Bedingung1) [{Anweisungen2;} break;}]
    [Anweisungen3;]
    if (Bedingung3) continue;]
    [Anweisungen4;]
}
```

Die for-in-Schleife

Diese Schleife listet alle Eigenschaften eines Objektes auf. Eines der wichtigsten Objekte ist `window`, es hat ca. 30 Eigenschaften, deren Namen und Werte auf diese Weise erfragbar sind. Der Test-Aufruf erfolgt in der Form:

```
for(i in window){alert("Eigenschaft "+i+" = "+window[i]);}
```

Gezielte Abbrüche von Schleifen mit `break` und `continue`

break

Allgemein wird eine Schleife beendet, wenn die überprüfte Bedingung nicht mehr erfüllt ist. Wie wir

gesehen haben, kann mit "break" eine Schleife sofort beendet, wenn die betreffende Stelle im Skript erreicht wird.

continue

Man kann damit an einer bestimmten Stelle unmittelbar den nächsten Schleifendurchlauf erzwingen und die nachstehenden Anweisungen innerhalb der Schleife ignorieren.

Funktionen

Mit Hilfe von Funktionen können Sie eigene, in sich abgeschlossene JavaScript-Anweisungen im Quelltext zusammenfassen, die Sie dann über den Aufruf der Funktion (Funktionsname) ausführen können. Dabei können Sie bestimmen, bei welchem Ereignis (zum Beispiel, wenn der Anwender einen Button anklickt) die Funktion aufgerufen und ihr Programmcode ausgeführt wird.

Beispiel:

```
function getFileName(filename) {  
    var name = filename.split('.');  
    return name[0] + "_copy." + name[1];  
}
```

Erläuterung:

Mit dem Schlüsselwort `function` leiten Sie die Definition einer Funktion ein. Dahinter folgt, durch ein Leerzeichen getrennt, ein frei wählbarer Funktionsname, im Beispiel: `getFileName`. Vergeben Sie einen Funktionsnamen, der das, was die Funktion leistet, ungefähr beschreibt. In dieser Funktion, wird dem übergeben Dateiname, der Namenszusatz `'_copy'` angefügt, somit bleibt die Ursprungsdatei für die weitere Bearbeitung unberührt. Diese Funktion kommt in dem nachstehenden Anwendungsbeispiel zum Einsatz.

try-catch Fehleranweisung

Diese Fehlerbehandlung erinnert an die if-else Anweisung. Der auftretende Fehler stellt bei try-catch die Bedingung dar, unter der ein alternativer Programmablauf im Skript eingeschlagen wird. JavaScript versucht zunächst, die Anweisungen im try-Block auszuführen. Ein im Skript gefundener Fehler wird an den catch-Block übergeben und alles was im catch-Block an Anweisungen vorgefunden wird, wird dort ausgeführt.

Beispiel:

```
try {  
    // Anweisungen  
}  
catch (error) {  
    alert(error);  
}
```

with-Anweisung

With erlaubt es, mehrere Anweisungen mit einem Objekt durchzuführen. Die with-Anweisung soll Schreibarbeit sparen und den Zugriff auf Objekt-Elemente übersichtlicher gestalten, d. h. Objektbandwürmer im Quellcode werden vermieden. Ich kann den Einsatz dieser Anweisung sehr empfehlen.

Erstes Anwendungsbeispiel

An dieser Stelle ist das Rüstzeug bereits vorhanden, um eigene JavaScripts zu erstellen. Es fehlen zwar noch die ganzen Informationen zu den Objekten in JavaScript in Verbindung mit Photoshop, aber von den syntaktischen Voraussetzungen her sind Sie jetzt dazu in der Lage.

Aufgabenstellung

Es sollen die Metadaten eines Bilddokumentes, die Informationen über Merkmale anderer Daten enthalten (Kameradaten, etc.) , mittels Skript, gelöscht, gespeichert oder geändert werden.

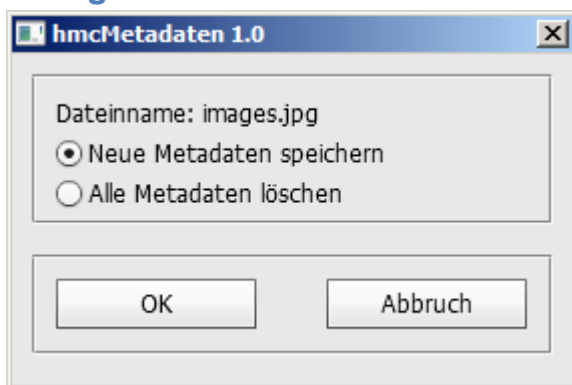
Hinweis:

Metadaten können in PS über 'Menü > Datei > **Dateiinformationen...**' angezeigt und editiert werden!

Programmbeschreibung

Es wird ein JavaScript erzeugt, das die "Benutzereinstellungen" die im Quelltext dieses Skriptes vom User definiert wurden in das aktuell geöffnete Bild, speichert oder löscht. Die Entscheidungsfindung, 'Neue Metadaten speichern', 'Alle Metadaten löschen' oder 'Abbruch' erfolgt über nachstehendes Dialogfenster. In der Titelzeile wird der Skriptname mit Versionshinweis und darunter der aktuelle Dateiname angezeigt.

Dialogfenster



Quelltext mit Beschreibung

Quelltext in JavaScript	Beschreibung
<pre>#target Photoshop #strict on var scriptName = "hmcMetadaten"; var scriptVersion = " 1.0";</pre>	Photoshop in den Vordergrund holen, Strikte Syntax-/Typisierungskontrolle einschalten Programmname und Version werden an Variablen übergeben um diese später oben in der Titelzeile im Dialogfeld darzustellen
<pre>var infoTitle = "Dokumenttitel"; var infoAuthor = "Hans Josef"; var infoAuthorPosition = "Firma"; var infoCaption = "Bild-Beschreibung"; var infoCaptionWriter = " Mustermann "; var infoKeywords = ["Eins", "Zwei", "Drei", "Vier"]; var copyrightedType = CopyrightedType.PUBLICDOMAIN; var copyrightNotice = "© 2013 Mustermann"; var ownerUrl = "www.meinewebsite.de";</pre>	Hier erfolgt die Festlegung der Metadaten durch den User. Die festgelegten Wert werden an die entsprechenden Variablen übergeben um diese später in die Bilddokument übertragen zu können. Hinweis: Diese Matadaten können in PS über Menü > Datei > Dateiinformationen... > Register: Beschreibung eingesehen oder editiert werden.
<pre>var infoCity = "Düsseldorf"; var infoProvinceState = "NRW"; var infoCountry = "Deutschland";</pre>	Dito, hier jedoch im Register: Ursprung

JavaScript-Grundlagen-Tutorial für Photoshop CS

<pre> var infoCredit = "keine Mitwirkende"; var infoSource = "Quellangaben"; var infoHeadline = "Meine Überschrift"; var infoInstructions = "keine Anweisungen"; var infoTransmissonReference = "keine Angabe"; </pre>	
<pre> try { </pre>	Hier beginnt die Fehlerbehandlung im try-Block
<pre> function getFileName(filename) { var name = filename.split('.'); return name[0] + "_copy." + name[1]; } </pre>	In dieser Funktion, wird dem übergeben Dateiname, der Namenszusatz '_copy' angefügt, somit bleibt die Ursprungsdatei für die weitere Bearbeitung unberührt.
<pre> function delateAllMetadata() { var width = app.activeDocument.width.value; var height = app.activeDocument.height.value; var resolution = app.activeDocument.resolution; var name = app.activeDocument.name; name = getFileName (name); var region = [[0,0],[0,width],[width,height],[width,0]]; app.activeDocument.selection.select (region); app.activeDocument.selection.copy(); app.activeDocument.selection.deselect(); documents.add (width, height, resolution, name, NewDocumentMode.RGB, DocumentFill.TRANSPARENT); app.activeDocument.paste(); purge (PurgeTarget.CLIPBOARDCACHE); } </pre>	<p>In dieser Funktion, werden alle Metadaten im aktiven Bilddokument gelöscht. Zunächst wird vom aktiven Bilddokument die Breite, Höhe, Auflösung, etc. an Variablen übergeben.</p> <p>Die Variable region ist ein Array mit vier x,y-Wertepaaren um einen Auswahlbereich um das aktive Bilddokument zu definieren. Es sind die x,y-Punkte: links-oben, rechts-oben, rechts-unten und links-unten. Dieser Auswahlbereich umfasst das komplette Bild. Jetzt wird dieser Bereich in die Zwischenablage kopiert und danach wird der Auswahlbereich aufgehoben.</p> <p>Nun wird ein neues Dokument erzeugt, dieses neue Dokument ist in Größe, Auflösung etc. vollkommen mit der Ursprungsdatei identisch, jedoch ohne Metadaten, Kameradaten etc. Das gespeicherte Bild in der Zwischenablage, wird nun deckungsgleich in das neu erzeugte Bilddokument eingefügt und anschließend wird die Zwischenablage entleert.</p>
<pre> function setImageInformation() { with (app.activeDocument) { info.title = infoTitle; info.author = infoAuthor; info.authorPosition = infoAuthorPosition; info.caption = infoCaption info.captionWriter = infoCaptionWriter; info.keywords = infoKeywords; info.copyrighted = copyrightedType; info.copyrightNotice = copyrightNotice; info.ownerUrl = ownerUrl; info.city = infoCity; info.provinceState = infoProvinceState; info.country = infoCountry; info.credit = infoCredit; info.source = infoSource; info.headline = infoHeadline; info.instructions = infoInstructions; info.transmissionReference = infoTransmissonReference; info.urgency.Urgency.NORMAL; } } </pre>	In dieser Funktion werden die im Skriptanfang festgelegten Matadaten, ins das neue Bilddokument gespeichert.

JavaScript-Grundlagen-Tutorial für Photoshop CS

<pre>function CreateDialog() { this.windowRef = null; var dlg = new Window("dialog", scriptName + scriptVersion, [100,100,380,265]); this.windowRef = dlg;</pre>	<p>In dieser Funktion wird das Dialogfenster erzeugt, referenziert und an die Variable dlg übergeben. In dieser Zeile wird ein neues Fenster-Objekt vom Typ 'dialog' angelegt und kommt als erster Eintrag in die runden Klammern für das neue Objekt. Es folgt die Angabe der Beschriftung der Titelzeile, diese wurden ja im Skriptanfang bereits an Variablen gebunden. Den Abschluss bilden die Positionsangaben/Abmessungen in Pixel. Das Objekt wird mit dlg erneut referenziert.</p>
<pre>dlg.infoPanel = dlg.add("panel", [10,10,270,85], ""); with (dlg.infoPanel) { infoFileName = add('statictext', [10,10,260,25], "Dateiname: " + app.activeDocument.name); rbtSaveNewData = add('radiobutton', [10,30,250,45], "Neue Metadaten speichern"); rbtSaveNewData.value = true; rbtClearMetaData = add('radiobutton', [10,50,250,65], "Alle Metadaten löschen"); }</pre>	<p>Hier wird ein Bereich 'panel' dem Dialogfenster hinzugefügt, dieser Typ 'panel' fasst die diversen Elemente wie Optionsfelder, Schaltflächen, Textfelder, etc. zusammen und umfasst die diversen Elemente mit einer Linie. Dieser Panel-Bereich gliedert somit das Dialogfenster. Der Erzeugungssyntax ist im wesentlichen mit dem Dialogfenster identisch. Ein Textfeld vom Typ 'statictext' mit dem statischen Text 'Dateiname:' und angefügtem ausgelesenen Dateinamen des aktiven Bildes, wird hinzugefügt. Zwei Optionsfelder vom Typ 'radiobutton' mit zugehörigem Text (Neue Metadaten speichern und Alle Metadaten löschen) werden im Panel entsprechend angeordnet.</p>
<pre>dlg.btnPanel = dlg.add("panel", [10,100,270,150], ""); with (dlg.btnPanel) { okBtn = dlg.btnPanel.add("button", [10,10,110,35], "OK"); cancelBtn = dlg.btnPanel.add("button", [145,10,245,35], "Abbruch");</pre>	<p>Hier wird ein weiterer Bereich 'panel' mit zwei Schaltflächen (OK und Abbruch) dem Dialogfenster hinzugefügt.</p>
<pre>okBtn.onClick = function() { if (rbtSaveNewData.value == true) { setImageInformation(); alert ("Neue Metadaten wurden gespeichert.", "Hinweis"); } if (rbtClearMetaData.value == true) { deleteAllMetadata(); alert ("Alle Metadaten wurden gelöscht.", "Hinweis"); } dlg.close(); } cancelBtn.onClick = function() { dlg.close(); }</pre>	<p>So richtig nützlich wird ein Dialogfenster erst, wenn die hier getroffenen Entscheidungen für den weiteren Programmablauf eine Auswertung liefern. In diesem Fall soll der Anwender eine der zwei Optionen auswählen und mit einem Klick auf die entsprechende Schaltfläche (OK oder Abbruch) bestätigen. Bei einem Klick auf OK, wird geprüft ob die Option (Neue Metadaten speichern) ausgewählt wurde, falls ja, wird die Funktion setImageInformation aufgerufen und anschließend wird in einem Dialogfenster die Aktion bestätigt. Bei einem Klick auf Abbrechen, wird das Dialogfeld geschlossen.</p>
<pre>// dlg.center(); dlg.frameLocation = [600,100]; dlg.show();</pre>	<p>An welcher Position soll das Dialogfenster im Bildschirm dargestellt werden? Hier erfolgt die Darstellung bei 600,100 Pixel (x,y-Koordinaten), d. h. die linke obere Ecke unseres Dialogfensters wird dort positioniert. In der untersten Zeile wird die Show-Methode für das Dialogfenster aufgerufen, d. h. erst jetzt ist das Dialogfenster sichtbar.</p>
<pre>function Cleanup() { app.preferences.rulerUnits = defaultRulerUnits;</pre>	<p>In dieser Funktion, werden die ursprünglich festgelegten Einheiten von Photoshop wieder zurückgestellt.</p>

JavaScript-Grundlagen-Tutorial für Photoshop CS

<pre>app.preferences.typeUnits = defaultTypeUnits; app.DisplayDialogs = defaultDisplayDialogs; return; }</pre>	
<pre>if (documents.length > 0) { try { app.bringToFront var defaultRulerUnits = app.preferences.rulerUnits; var defaultTypeUnits = app.preferences.typeUnits; var defaultDisplayDialogs = app.DisplayDialogs; if (defaultRulerUnits != Units.PIXELS){ preferences.rulerUnits = Units.PIXELS; } CreateDialog(); } catch(e) { if (e.line != undefined) alert(scriptName + " Zeile[" + e.line + "]" + e); else alert(scriptName + " " + e); Cleanup(); } } else { alert(scriptName + " Es ist kein Bilddokument geöffnet!"); }</pre>	<p>Hier steht das Hauptprogramm bzw. hier erfolgt der eigentliche Programmstart. Zunächst wird überprüft, ob überhaupt ein Bilddokument geöffnet wurde, falls ja, wird Photoshop in Vordergrund geholt. Anschließend werden die vorhandenen Einstellungen gespeichert und die für diese Anwendung notwendigen Einheiten in Photoshop eingestellt.</p> <p>Die zuvor beschriebene Funktion CreateDialog wird aufgerufen.</p> <p>Im catch-Block werden eventuelle Fehler angezeigt. Zum Beispiel erscheint ein Warnungsdialogfenster, wenn kein Bilddokument geöffnet wurde:</p> <p>hmcMetadaten Es ist kein Bilddokument geöffnet!</p>
<pre>scriptName = null; scriptVersion = null; width = null; height = null; resolution = null;</pre>	Den Speicher des PC's wieder freimachen.
<pre>} // try Ende catch (e) { alert (e.description, "Fehler!"); }</pre>	Eventuell auftretende Fehler, werden in diesem Fehlerblock (catch-Block) abgefangen und angezeigt.

Anhang

Download

Der komplette Quellcode von **hmcMatadaten.jsx** kann als ZIP-Datei heruntergeladen werden. Dies ist aus meiner Sicht nützlich, weil gerade am Anfang, noch häufiger Schreibfehler oder sachliche Fehler, gemacht werden. Wenn JavaScript nicht das tut was es soll, ist die Enttäuschung groß und die Ablehnung steigert sich, was wir ja nicht brauchen. Ich kann also versichern, dieses Skript hat - schon mal - funktioniert.

Lizenz

Dieses Tutorial ist für die Nutzung gedacht, d. h. Anwenden, Experimentieren, Lesen, Verstehen, sich freuen wenn alles klappt, Verlinken, Kopieren und Verteilen. Nicht darunter fällt das Zerpfücken und

Verändern dieses Tutorial, wer das möchte, soll doch bitte selber eins erstellen. Vielen Dank für Ihre Einsicht!