

# **ADOBE<sup>®</sup> SKRIPTE-EINFÜHRUNG**

© Copyright 2012 Adobe Systems Incorporated. Alle Rechte vorbehalten.

### *Adobe® Skripte-Einführung*

HINWEIS: Alle hierin enthaltenen Informationen sind Eigentum von Adobe Systems Incorporated. Diese Publikation darf ohne vorherige schriftliche Genehmigung von Adobe Systems Incorporated weder ganz noch teilweise auf elektronische oder mechanische Weise, durch Fotokopieren, Aufzeichnen oder auf andere Weise, weder in Papierform noch in elektronischer Form vervielfältigt oder übertragen werden. Die in diesem Dokument beschriebene Software wird im Rahmen einer Lizenz zur Verfügung gestellt und darf nur gemäß den Lizenzbedingungen genutzt oder kopiert werden.

Diese Publikation und die darin enthaltenen Informationen sind ohne Gewähr, können jederzeit ohne Vorankündigung geändert werden und dürfen nicht als bindende Verpflichtung von Adobe Systems Incorporated ausgelegt werden. Adobe Systems Incorporated ist nicht verantwortlich oder haftbar für Fehler oder Ungenauigkeiten, übernimmt keinerlei ausdrückliche, stillschweigende oder gesetzliche Gewährleistung für diese Publikation und schließt ausdrücklich jegliche Gewährleistung hinsichtlich Marktfähigkeit, Eignung für bestimmte Zwecke und Nichtverletzung von Rechten Dritter aus.

Alle in den Abbildungen erwähnten Firmennamen dienen lediglich zur Veranschaulichung und nehmen keinerlei Bezug auf tatsächlich bestehende Unternehmen.

Adobe, das Adobe-Logo, Creative Suite, Illustrator, InDesign und Photoshop sind entweder eingetragene Marken oder Marken von Adobe Systems Incorporated in den Vereinigten Staaten und/oder anderen Ländern.

Apple, Mac OS und Macintosh sind in den Vereinigten Staaten und anderen Ländern eingetragene Marken von Apple Computer, Inc. Microsoft und Windows sind entweder eingetragene Marken oder Marken der Microsoft Corporation in den Vereinigten Staaten und anderen Ländern. JavaScript und alle Bezeichnungen im Zusammenhang mit Java sind Marken oder eingetragene Marken von Sun Microsystems, Inc. in den Vereinigten Staaten und anderen Ländern. UNIX® ist eine eingetragene Marke von The Open Group.

Alle anderen Marken sind Eigentum ihrer jeweiligen Inhaber.

Wenn dieses Handbuch mit Software vertrieben wird, die eine Endbenutzer-Lizenzvereinbarung umfasst, werden dieses Handbuch sowie die darin beschriebene Software unter Lizenz bereitgestellt und dürfen nur entsprechend den Bedingungen der Lizenz verwendet oder vervielfältigt werden. Kein Teil dieser Dokumentation darf ohne vorherige schriftliche Genehmigung von Adobe Systems Incorporated reproduziert, in Datenbanken gespeichert oder in irgendeiner Form – elektronisch, mechanisch, auf Tonträgern oder auf irgendeine andere Weise – übertragen werden, es sei denn, die Lizenz gestattet dies ausdrücklich. Beachten Sie, dass der Inhalt dieses Handbuchs urheberrechtlich geschützt ist, auch wenn er nicht zusammen mit Software vertrieben wird, die eine Endbenutzer-Lizenzvereinbarung umfasst.

Der Inhalt dieses Handbuchs dient lediglich Informationszwecken, kann jederzeit ohne Vorankündigung geändert werden und stellt keinerlei Verpflichtung seitens Adobe Systems Incorporated dar. Adobe Systems Incorporated übernimmt keine Verantwortung oder Haftung für Fehler oder Ungenauigkeiten in den in diesem Handbuch enthaltenen Informationen.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

# Inhalt

<b>1</b>	<b>Einführung</b> .....	<b>5</b>
	Ist das Arbeiten mit Skripten schwer zu erlernen? .....	5
	Gründe für den Einsatz von Skripten .....	5
	Einsatzmöglichkeiten für Skripte .....	5
	Aktionen und Makros .....	6
	Was genau sind Skripte? .....	6
	AppleScript .....	6
	VBScript .....	7
	JavaScript .....	7
	Erste Schritte .....	8
	AS .....	8
	JS .....	8
	VBS .....	8
<b>2</b>	<b>Skriptgrundlagen</b> .....	<b>9</b>
	Skriptbausteine .....	9
	Objekte, Eigenschaften, Methoden und Befehle .....	9
	Objekte verwenden .....	9
	DOM-Konzepte .....	10
	Variablen .....	11
	Objektverweise .....	11
	Variablen .....	12
	Variablen benennen .....	13
	Objektkollektionen oder Elemente als Objektverweise .....	13
	Nummerierung von Objekten durch Elemente und Kollektionen .....	14
	Verweise auf das aktuelle oder aktive Objekt .....	15
	Eigenschaften verwenden .....	16
	AS .....	18
	JS .....	19
	VBS .....	20
	Schreibgeschützte Eigenschaften und Lese-/Schreibeigenschaften .....	20
	Warnfelder zum Anzeigen eines Eigenschaftswerts verwenden .....	20
	Konstantenwerte und Aufzählungen .....	22
	AS .....	22
	JS .....	23
	VBS .....	24
	Variablen für Eigenschaftswerte verwenden .....	24
	Methoden oder Befehle verwenden .....	24
	Befehls- oder Methodenparameter .....	25
	Erforderliche Parameter .....	25
	Mehrere Parameter .....	26
	Tell-Anweisungen (nur AS) .....	27

Hinweise über Variablen .....	28
Variablenwert ändern .....	28
Variablen verwenden, um auf vorhandene Objekte zu verweisen .....	29
Skriptdateien lesbar gestalten .....	29
Skript kommentieren .....	29
Fortlaufende lange Zeilen in AppleScript und VBScript .....	30
Arrays verwenden .....	31
Objekte erstellen .....	32
Weitere Informationen über Skripte .....	32
<b>3    Eigenschaften von Objekten und Methoden .....</b>	<b>33</b>
Skriptumgebungs-Browser verwenden .....	33
AppleScript-Datenfunktionsverzeichnisse .....	33
AppleScript-Funktionsverzeichnisse anzeigen .....	33
AppleScript-Funktionsverzeichnisse verwenden .....	33
JavaScript-Objektmodell-Viewer .....	36
VBScript-Typbibliotheken .....	36
VBScript-Typbibliotheken .....	36
VBScript-Typbibliotheken verwenden .....	37
Adobe-Skriptreferenzen verwenden .....	41
Mit der Elementtabelle eines Objekts arbeiten (nur AS) .....	42
Mit der Eigenschaftstabelle eines Objekts arbeiten .....	42
Mit der Methodentabelle von Objekten arbeiten .....	44
<b>4    Erweiterte Skripttechniken .....</b>	<b>46</b>
Konditionalanweisungen .....	46
if-Anweisungen .....	46
if else -Anweisungen .....	47
Schleifen .....	48
Weitere Informationen über Skripte .....	49
<b>5    Fehlerbehebung .....</b>	<b>50</b>
Reservierte Wörter .....	50
Fehlermeldungen des AppleScript-Skripteditors .....	50
ESTK-Fehlermeldungen .....	51
VBScript-Fehlermeldungen .....	52
<b>6    Bibliographie .....</b>	<b>53</b>
AppleScript .....	53
JavaScript .....	53
VBScript .....	53
<b>Index .....</b>	<b>54</b>

# 1 Einführung

Skripte sind leistungsfähige Werkzeuge, mit denen Sie zahlreiche Funktionen von vielen Adobe® Creative Suite®-Anwendungen steuern und automatisieren können. Sie können mit Skripten Ihre Arbeit erheblich schneller und einfacher erledigen und Ihre Arbeitsabläufe völlig neu gestalten.

## Ist das Arbeiten mit Skripten schwer zu erlernen?

Verwechseln Sie das Arbeiten mit Skripten nicht mit dem Programmieren. Sie müssen weder Informatiker noch Mathematiker sein, um einfache Skripte zu schreiben, die eine breite Palette von Routineaufgaben automatisieren.

Jedes Skriptelement entspricht einem Werkzeug oder einer Palette bzw. einem Menüelement in einer Adobe-Anwendung. Durch Ihre Arbeit mit Adobe kennen Sie bereits alle Skriptelemente. Wenn Sie wissen, welche Aktionen Sie mit den Adobe-Anwendungen durchführen möchten, werden Sie schnell lernen, wie Sie Skripte schreiben.

## Gründe für den Einsatz von Skripten

Die Kreativität steht bei Ihrer Arbeit im Vordergrund. Leider gehören dazu auch viele Aufgaben, die sehr unkreativ sind. Vermutlich verbringen Sie viel Zeit damit, immer wieder gleiche oder ähnliche Arbeitsschritte durchzuführen.

Ein Assistent könnte Ihnen viele der stereotypen Aufgaben abnehmen. Jemand, der Ihre Anweisungen perfekt und mit vorhersehbarer Genauigkeit ausführt, der jederzeit verfügbar ist, seine Arbeiten umgehend erledigt und keine Kosten verursacht.

Skripte können die Aufgaben eines solchen Assistenten übernehmen. Sie müssen nicht viel Zeit investieren, um einfache Routineaufgaben, die sehr zeitraubend sein können, durch Skripte ausführen zu lassen. Moderne Skriptsprachen sind einfach zu erlernen; dennoch können Sie mit ihrer Hilfe auch äußerst anspruchsvolle Aufgaben durchführen. Mit zunehmenden Skript-Kenntnissen können Sie zu komplexeren Skripten übergehen, die für Sie weiterarbeiten, auch wenn Sie nicht anwesend sind.

## Einsatzmöglichkeiten für Skripte

Gehören zu Ihrem Arbeitsablauf Routineaufgaben, die Sie viel Zeit kosten? Diese Aufgaben können Sie durch Skripte ausführen lassen. Überlegen Sie sich Folgendes:

- ▶ Welche Schritte sind erforderlich, um die Aufgabe durchzuführen?
- ▶ Unter welchen Bedingungen führen Sie die Aufgabe durch?

Sobald Sie den Vorgang erfasst haben, den Sie für die Aufgabe manuell ausführen, können Sie ihn durch ein Skript darstellen.

## Aktionen und Makros

Wenn Sie bereits mit Aktionen und Makros vertraut sind, haben Sie in etwa einen Eindruck von der Effizienz von Skripten. Skripte sind jedoch Aktionen oder Makros überlegen, weil Sie mit Skripten mehrere Dokumente und mehrere Anwendungen innerhalb eines einzigen Skripts steuern können. Sie können beispielsweise ein Skript schreiben, das ein Bild in Adobe Photoshop<sup>®</sup> verändert und Adobe InDesign<sup>®</sup> anweist, dieses Bild zu integrieren.

Außerdem kann das Skript auf intelligente Art Daten abrufen und auf Daten antworten. Beispiel: Sie haben ein Dokument mit Fotos verschiedener Größe. Sie können ein Skript schreiben, das die Größe der einzelnen Fotos berechnet und Rahmen mit verschiedenen Farben auf der Basis der Größe erstellt, so dass Symbole blaue, kleine Illustrationen grüne und halbseitige Bilder silberne Rahmen haben.

Sie können innerhalb der Anwendung auch Aktionen durch Skripte ausführen lassen.

## Was genau sind Skripte?

Ein Skript enthält eine Reihe von Anweisungen, die eine Anwendung zur Durchführung von bestimmten Aufgaben veranlassen.

Die Anweisungen müssen in einer Sprache geschrieben werden, die von den Anwendungen verstanden wird. Skriptfähige Adobe-Anwendungen unterstützen mehrere Skriptsprachen.

Unter Mac OS<sup>®</sup> stehen die folgenden Optionen zur Verfügung:

- ▶ AppleScript
- ▶ JavaScript

Unter Windows<sup>®</sup> stehen die folgenden Optionen zur Verfügung:

- ▶ VBScript (möglich sind auch Visual Basic und VBA)
- ▶ JavaScript

Die folgenden kurzen Beschreibungen helfen Ihnen dabei, die optimale Sprache zu wählen.

## AppleScript

AppleScript ist eine „einfache“ Skriptsprache, die von Apple entwickelt wurde. Sie gilt als die einfachste Skriptsprache.

Um AppleScript-Skripte zu schreiben, können Sie den Skripteditor von Apple verwenden, der sich bei der Standardinstallation von Mac OS unter

*Systemlaufwerk:Programme:AppleScript:Script Editor* befindet.

Weitere Informationen über den Skripteditor finden Sie in der Skripteditor-Hilfe.

## VBScript

VBScript ist eine herunterskalierte Version der Visual Basic-Programmiersprache, die von Microsoft entwickelt wurde. VBScript kommuniziert mit den Host-Anwendungen über ActiveX-Skripte. VBScript ist die Visual Basic-Sprache, die offiziell durch CS6 unterstützt wird. Sie können auch Skripte in VBA und Visual Basic selbst schreiben.

Im Internet finden Sie verschiedene gute VBScript-Editoren. Wenn Sie Microsoft Office-Anwendungen installiert haben, können Sie auch den integrierten Visual Basic-Editor verwenden, indem Sie **Extras > Makro > Visual Basic-Editor** wählen.

## JavaScript

JavaScript ist eine sehr gebräuchliche Skriptsprache, die ursprünglich entwickelt wurde, um Webseiten interaktiv zu gestalten. Ähnlich wie AppleScript ist auch JavaScript einfach zu lernen.

**HINWEIS:** Adobe hat eine erweiterte Version von JavaScript mit dem Namen ExtendScript entwickelt, mit der Sie bestimmte Adobe-Werkzeuge und Skriptfunktionen nutzen können. Wenn Sie gerade anfangen, sich mit Skriptsprachen zu befassen, sind die Unterschiede zwischen diesen beiden Sprachen für Sie ohne Belang. Sie sollten jedoch die JavaScript-Skripte mit der Endung `.jsx` statt mit der herkömmlichen Endung `.js` erstellen.

JavaScript bietet ein paar Vorteile gegenüber AppleScript und Visual Basic:

- ▶ Sie können die Skripte sowohl unter Windows als auch unter Mac OS verwenden. Wenn Sie die Skripte auf beiden Plattformen verwenden oder freigeben möchten, sollten Sie JavaScript verwenden.
- ▶ In Adobe Illustrator® und InDesign können Sie auf Skripte in einer der unterstützten Sprachen aus der Anwendung heraus zugreifen. In Photoshop können Sie jedoch aus der Anwendung nur auf `.jsx`-Dateien zugreifen. Sie müssen AppleScript- oder Visual Basic-Skripte außerhalb der Anwendung ausführen. Dies ist kein schwerwiegender Nachteil; für die Ausführung der Skripte sind jedoch einige zusätzliche Mausklicks erforderlich.
- ▶ Wenn Sie möchten, dass `.jsx`-Skripte automatisch beim Öffnen der Anwendung ausgeführt werden, speichern Sie sie im Ordner Startup Scripts der Anwendung. Informationen über die Ordner Startup Scripts finden Sie im Skripthandbuch der Anwendung.

Um Skripte in JavaScript zu schreiben, können Sie jeden beliebigen Texteditor verwenden. Sie können auch das ESTK (ExtendScript Tool Kit) verwenden, das zum Lieferumfang der Adobe-Anwendungen gehört. Das ESTK ist durch seine vielen Funktionen benutzerfreundlicher als ein Texteditor. Es umfasst eine integrierte Syntaxprüfung, die Probleme in den Skripten erkennt und erklärt, wie sie zu beheben sind. Skripte können außerdem direkt aus dem ESTK ausgeführt werden, ohne die Datei zu speichern. Durch diese zweite Funktion können Sie viel Zeit sparen, besonders am Anfang, wenn Sie ein Skript mehrmals testen und bearbeiten müssen, bevor es richtig funktioniert.

Bei einer Adobe-Standardinstallation befindet sich das ESTK an folgendem Speicherort:

Mac OS: *System laufwerk*:Programme:Dienstprogramme:Adobe Utilities - CS6:ExtendScript Toolkit CS6

Windows: *Laufwerk*:/Programme/Adobe/Adobe Utilities - CS6/ExtendScript Toolkit CS6

Einzelheiten finden Sie im *JavaScript Tools Guide*.

## Erste Schritte

Sie können nun mit dem Schreiben des ersten Skripts beginnen.

**HINWEIS:** Wenn Sie mit dem Ausführen der Skripte Probleme haben, finden Sie weitere Informationen in [Kapitel 5, „Fehlerbehebung“](#).

### AS

1. Öffnen Sie den Skripteditor und geben Sie Folgendes ein (wobei Sie die Adobe-Anwendung in Anführungszeichen ersetzen):

```
tell application "Adobe Photoshop CS6"  
    make document  
end tell
```

2. Klicken Sie auf **Ausführen**.

### JS

1. Öffnen Sie das ESTK und wählen Sie eine Anwendung aus der Dropdown-Liste in der unteren linken Ecke eines Dokumentenfensters aus.
2. Geben Sie in die Palette der JavaScript Console Folgendes ein:

```
app.documents.add()
```

3. Führen Sie einen der folgenden Schritte durch:

- ▶ Klicken Sie im Fenster „Document“ oben in der Symbolleiste auf das Symbol „Run“.
- ▶ Klicken Sie auf **F5**.
- ▶ Wählen Sie **Debug -> Run**.

### VBS

1. Geben Sie Folgendes in einem Texteditor ein (wobei Sie eine Adobe-Anwendung in Anführungszeichen in der zweiten Zeile ersetzen):

```
Set appRef = CreateObject("Photoshop.Application")  
appRef.Documents.Add()
```

2. Speichern Sie die Datei als Textdatei mit einer .vbs-Erweiterung (beispielsweise create\_doc.vbs).
3. Doppelklicken Sie auf die Datei in Windows Explorer.



## 2 Skriptgrundlagen

In diesem Kapitel werden die grundlegenden Konzepte von Skripten sowohl unter Windows als auch unter Mac OS beschrieben. Produktspezifische Anweisungen finden Sie im Skripthandbuch der Adobe-Anwendung.

### Skriptbausteine

Das erste Skript, das ein neues Dokument erstellt hat, war wie ein englischer Satz konstruiert – mit einem Substantiv (`document`) und einem Verb (`make` in AS, `add()` in JS und `Add` in VBS). In Skripten wird ein Substantiv *Objekt* und ein Verb *Befehl* (in AS) oder *Methode* genannt (in JS und VBS).

So wie Sie ein Substantiv durch Adjektive modifizieren können, können Sie ein Skriptobjekt durch *Eigenschaften* modifizieren. Um einen Befehl oder eine Methode zu modifizieren, verwenden Sie *Parameter*.

### Objekte, Eigenschaften, Methoden und Befehle

Wenn Sie eine Adobe-Anwendung verwenden, öffnen Sie eine Datei oder ein Dokument und erstellen oder ändern Ebenen, Text, Rahmen, Kanäle, Grafiklinien und andere Design-Elemente. Bei diesen Dingen handelt es sich um Objekte.

Um eine Skriptanweisung zu schreiben, können Sie ein Objekt erstellen oder auf ein vorhandenes Objekt verweisen und anschließend einen der folgenden Schritte ausführen:

- ▶ Legen Sie Werte für die Objekteigenschaften fest. Sie können beispielsweise den Namen, die Länge oder die Breite eines Dokuments festlegen. Sie können den Namen, die Farbe oder die Deckkraft festlegen.
- ▶ Legen Sie Befehle oder Methoden fest, um dem Skript mitzuteilen, welche Aktionen mit den Objekten durchgeführt werden sollen. Sie können beispielsweise ein Dokument öffnen, schließen, speichern und drucken. Sie können eine Ebene zusammenführen, verschieben oder rastern.

Beim Schreiben von Skripten ist allerdings darauf zu achten, dass nur die Eigenschaften oder Methoden/Befehle verwendet werden, die für das Objekt erlaubt sind. Woher wissen Sie, welche Eigenschaften und Methoden für welches Objekt erlaubt sind? In den meisten Fällen ist es eine Sache der Logik. Wenn Sie eine Einstellung in der Adobe-Anwendung festlegen können, ist dies in der Regel auch in einem Skript möglich.

Sie finden zu diesem Thema jedoch auch ausführliche Informationen in Skript-Ressourcen, in denen Sie alle Informationen finden, um Skriptobjekte zu erstellen, definieren und ändern. Informationen, wie Sie diese Ressourcen finden und verwenden, finden Sie in [Kapitel 3, „Eigenschaften von Objekten und Methoden“](#).

### Objekte verwenden

Beim Verwenden von Objekten in Skripten ist es vor allen Dingen wichtig zu wissen, wie auf Objekte verwiesen wird. Wie können Sie der Anwendung vermitteln, welches Objekt das Skript ändern soll? Sie können in der Benutzeroberfläche der Anwendung das Objekt auswählen, indem Sie es anklicken. In einem Skript ist dies nicht so einfach.

## DOM-Konzepte

Skriptsprachen verwenden manchmal ein *Dokument-Objektmodell* (DOM), um Objekte so zu organisieren, dass sie leicht zu erkennen sind. Hinter einem DOM steht das Prinzip der *Container-Hierarchie*. Objekte auf erster Ebene *enthalten* die Objekte der nächsten Ebene, die wiederum eine weitere Ebene von Objekten enthalten usw.

Das Objekt der obersten Ebene in einem beliebigen DOM einer Adobe-Anwendung ist beispielsweise das Anwendungsobjekt. Es folgen Dokumentobjekte, die alle anderen Objekte wie Ebenen, Kanäle, Seiten, Textrahmen usw. enthalten. Diese Objekte können Objekte enthalten, die Dokumente nicht direkt enthalten können. Beispielsweise kann in InDesign oder Illustrator ein Textrahmen Wörter enthalten. Ein Dokument kann nur dann Wörter enthalten, wenn es über einen Textrahmen verfügt. Auch in Photoshop kann ein Dokument eine Ebene enthalten und eine Ebene einen Textrahmen. Ein Dokument kann jedoch nur dann einen Textrahmen enthalten, wenn das Dokument eine Ebene enthält.

**HINWEIS:** Ein Objekt, das ein Objekt enthält, wird auch *übergeordnetes* Objekt genannt.

Im ersten Skript haben Sie zunächst das Anwendungsobjekt benannt (oder es im ESTK ausgewählt) und anschließend das Dokument innerhalb der Anwendung erstellt. Wenn Sie im nächsten Schritt eine Ebene erstellen möchten, muss das Skript das Dokument bezeichnen, in dem Sie die Ebene erstellen möchten. Wenn das Skript der Anwendung nicht mitteilt, wo das Objekt genau erstellt werden soll, kann es nicht erfolgreich ausgeführt werden.

**HINWEIS:** Ein Diagramm des DOM für eine bestimmte Anwendung finden Sie im Skripthandbuch der Anwendung.

Im folgenden Abschnitt wird beschrieben, wie Sie mit dem DOM-Prinzip eine Ebene zu einem Dokument hinzufügen. (Beachten Sie, wenn Sie dieses Skript für Photoshop verwenden, dass eine Ebene in AS `art layer` genannt wird. Ebenen werden in JS `artLayers` bzw. in VBS `ArtLayers` genannt).

```
AS tell application "Adobe Illustrator CS6"
    make document
    make layer in document
end tell
```

```
JS app.documents.layers.add()
```

```
VBS Set appRef = CreateObject("Illustrator.Application")
docRef.Documents.Add
appRef.Documents.Layers.Add
```

Beim Ausführen dieser Skripte wird eine Fehlermeldung angezeigt, da die Anwendung nicht erkennt, welches Dokument gemeint ist. Vermutlich ist nur ein Dokument geöffnet; dies ist jedoch nicht immer der Fall. Daher gibt es für Skriptsprachen die strenge Anweisung, dass alle Objekte in jeder Skriptanweisung explizit bezeichnet werden.

In diesem Handbuch werden drei Arten vorgestellt, wie auf Objekte verwiesen wird:

- ▶ Variablen
- ▶ Kollektions- oder Elementnummern
- ▶ Das „aktuelle“ Objekt oder die „aktive“ Objekteigenschaft

## Variablen

Eine *Variable* ist ein von Ihnen erstelltes Element in einem Skript, das Daten enthält. Bei den Daten, auch *Wert* der Variablen genannt, kann es sich um ein Objekt im Skript oder eine Eigenschaft handeln, die ein Objekt beschreibt. Eine Variable ist mit einem Aliasnamen vergleichbar, den Sie einem Objekt oder anderen Daten geben.

Indem Sie eine Variable verwenden, die ein Objekt enthält, wird der Verweis auf dieses Objekt erleichtert. Die meisten Skriptautoren erstellen eine Variable für jedes Objekt im Skript.

Durch die folgenden Skripte wird ein Dokument erstellt, wie bei Ihrem ersten Skript. Durch diese Version des Skripts wird eine Variable mit dem Namen `myDoc` erstellt, die das Dokument enthält. Vergleichen Sie diese Skripte mit dem ersten Skript. (Siehe [„Erste Schritte“ auf Seite 8.](#))

**AS** Um eine Variable in AS zu erstellen, verwenden Sie den Befehl `set`, gefolgt vom Variablennamen. Um der Variable einen Datenwert zuzuweisen, verwenden Sie `to`, gefolgt vom Wert.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
end tell
```

**JS** Um eine Variable in JS zu erstellen, verwenden Sie den Befehl `var`, gefolgt vom Variablennamen. Um einen Datenwert zuzuweisen, verwenden Sie ein Gleichheitszeichen (`=`), gefolgt vom Wert. Leerzeichen auf beiden Seiten des Gleichheitszeichens spielen keine Rolle.

```
var myDoc = app.documents.add()
```

**VBS** Um eine Variable in VBS zu erstellen, verwenden Sie den Befehl `Set`, gefolgt vom Variablennamen. Um einen Datenwert zuzuweisen, verwenden Sie ein Gleichheitszeichen (`=`), gefolgt vom Wert. Leerzeichen auf beiden Seiten des Gleichheitszeichens spielen keine Rolle.

```
Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.Documents.Add
```

## Objektverweise

Nachdem Sie nun eine Möglichkeit haben, auf das im Skript erstellte Objekt `document` zu verweisen, können Sie die Ebene problemlos hinzufügen. (Beachten Sie, wenn Sie dieses Skript für Photoshop verwenden, dass eine Ebene in AS `art layer` genannt wird. Ebenen werden in JS `artLayers` bzw. in VBS `ArtLayers` genannt).

**AS**

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    make layer in myDoc
end tell
```

Wir erstellen nun eine weitere Variable, die die Ebene enthält. Auf diese Weise können wir auf die Ebene verweisen, wenn wir ihre Eigenschaften definieren oder der Ebene ein Objekt hinzufügen möchten.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc
end tell
```

```
JS      var myDoc = app.documents.add()
         myDoc.layers.add()
```

Wir verwenden dasselbe Skript und erstellen nun eine Variable, die die Ebene enthält.

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
```

```
VBS    Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.Documents.Add
docRef.Layers.Add
```

Wir verwenden dasselbe Skript und erstellen nun eine Variable, die die Ebene enthält.

```
Set appRef = CreateObject("Photoshop.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.Layers.Add
```

## Variablen

Variablen, die Objekte enthalten, enthalten außerdem die gesamte Container-Hierarchie, die das Objekt definiert. Wenn Sie beispielsweise auf `myLayer` verweisen möchten, müssen Sie nicht auf das Dokument verweisen, das die Ebene enthält. Die folgenden Skripte erstellen einen Textrahmen in `myLayer`. Wenn Sie `myLayer` verwenden, müssen Sie keine Daten über die Container-Hierarchie der Ebene angeben.

**HINWEIS:** Im folgenden Skript wird die Eigenschaft `contents` verwendet, um Text zum Rahmen hinzuzufügen. Im Moment möchten wir uns mit der Verwendung von Eigenschaften noch nicht beschäftigen.

Das folgende Skript verwendet Objekte und Eigenschaften, die im Illustrator CS6-Objektmodell definiert sind. Daher funktioniert es z. B. nicht in InDesign oder Photoshop.

```
AS      tell application "Adobe Illustrator CS6"
         set myDoc to make document
         set myLayer to make layer in myDoc
         set myTextFrame to make text frame in myLayer
         set contents of myTextFrame to "Hello world!"
end tell
```

```
JS      var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
var myTextFrame = myLayer.textFrames.add()
myTextFrame.contents = "Hello world!"
```

```
VBS    Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.Layers.Add
Set frameRef = layerRef.TextFrames.Add
myTextFrame.Contents = "Hello world!"
```

## Variablen benennen

Die Skripte sind leichter zu lesen, wenn Sie beschreibende Namen für die Variablen wählen. Variablenamen wie `x` oder `c` sind nicht hilfreich, wenn Sie ein Skript erneut lesen. Verwenden Sie Namen, die Aufschluss geben über die enthaltenen Daten der Variablen, wie `theDocument` oder `myLayer`.

Wenn Sie die Variablen benennen, können Sie durch ein Standardpräfix Variablen von Objekten, Befehlen und Schlüsselwörtern des Skriptsystems unterscheiden. Beispiel:

- ▶ Sie können das Präfix „doc“ am Beginn aller Variablen verwenden, die Objekte vom Typ `Document` enthalten, wie `docRef` oder „layer“, um die Variablen zu bezeichnen, die Objekte vom Typ `Art Layer` enthalten, wie `layerRef` und `layerRef2`.
- ▶ Sie können das Präfix „my“ verwenden, um persönliche Elemente hinzuzufügen, die Ihre Variablen von Skriptobjekten unterscheiden. Beispiele: `myDoc` oder `myLayer` oder `myTextFrame`.

Alle Variablenamen müssen den folgenden Richtlinien entsprechen:

- ▶ Variablenamen dürfen aus nur einem Wort bestehen (keine Leerzeichen). Häufig werden die interne Großschreibung (wie `myFirstPage`) oder Unterstriche verwendet (`my_first_page`), um die Lesbarkeit der Namen zu verbessern. Die Variablenamen dürfen nicht mit einem Unterstrich beginnen.
- ▶ Variablenamen können Ziffern enthalten. Sie dürfen jedoch nicht mit einer Ziffer beginnen.
- ▶ Variablenamen dürfen keine Anführungszeichen oder Satzzeichen mit Ausnahme von Unterstrichen enthalten.
- ▶ Bei Variablenamen in JavaScript und VBScript muss die Groß- und Kleinschreibung beachtet werden. `thisString` ist nicht gleichbedeutend mit `thisstring` oder `ThisString`. Bei Variablenamen in AppleScript muss die Groß-/Kleinschreibung nicht beachtet werden.
- ▶ Jeder Variable in Ihrem Skript muss ein eindeutiger Name zugeordnet werden.

## Objektkollektionen oder Elemente als Objektverweise

Skriptsprachen platzieren jedes Objekt in einer *Kollektion* (JS oder VBS) oder einem *Element* (AS) und weisen dann die Objektnummer, *Index*, genannt, innerhalb des Elements oder der Kollektion zu. Die Objekte in einem Element oder einer Kollektion sind identische Objekttypen. Beispiel: Jedes `channel`-Objekt in Ihrem Dokument gehört zu einem `channels`-Element oder einer `channels`-Kollektion; jedes `art layer`-Objekt gehört zu einem `art layers`-Element oder einer `artLayers`-Kollektion.

Auf Deutsch können Sie auf ein Dokument verweisen, indem Sie sagen: „Geben Sie mir das erste Dokument in der Sammlung.“ Auch in Skriptsprachen können Sie ein Objekt in ähnlicher Weise bezeichnen, indem Sie seinen Element- oder seinen Kollektionsnamen sowie seinen Index verwenden.

- ▶ In AS verweisen Sie auf das erste Dokument im Element `document` als `document 1`.
- ▶ In JS ist `documents [0]` (beachten Sie die eckigen Klammern um den Index) das erste Dokument, da – was zunächst schwer zu merken ist – in JavaScript die Nummerierung von Kollektionsobjekten bei 0 beginnt.
- ▶ In VBS ist das erste Dokument `Documents (0)` (beachten Sie die Klammern um den Index). VBS beginnt mit der Nummerierung von Kollektionsobjekten bei 1.

Die folgenden Skripte verweisen auf die Objekte `document` und `layer` nach Index, um neue Objekte hinzuzufügen.

**HINWEIS:** Da das folgende Skript keine Variablen verwendet, ist die vollständige Container-Hierarchie in jedem Objektverweis erforderlich. In der Anweisung, die eine Ebene hinzufügt, muss das Skript z. B. das Dokument bezeichnen, zu dem die Ebene hinzugefügt wird. Um der Ebene einen Textrahmen hinzuzufügen, muss das Skript den Index nicht nur für die Ebene angeben, die den Rahmen enthält, sondern auch das Dokument bezeichnen, das die Ebene enthält.

```
AS tell application "Adobe InDesign CS6"
    make document
    make layer in document 1
    make text frame in layer 1 of document 1
end tell
```

**HINWEIS:** Skriptanfänger, die AppleScript verwenden, sollten keine Elementnummern als Objektreferenzen verwenden, wenn das Element mehr als ein Objekt enthält. Einzelheiten dazu finden Sie in [„Nummerierung von Objekten durch Elemente und Kollektionen“ auf Seite 14.](#)

**JS** In JavaScript bezeichnen Sie den Index eines Elements, indem Sie den Kollektionsnamen gefolgt durch den Index in eckigen Klammern verwenden ([ ]).

```
app.documents.add()
app.documents[0].layers.add()
app.documents[0].layers[0].textFrames.add()
```

**HINWEIS:** Denken Sie daran, dass Indexnummern in JS innerhalb einer Kollektion bei 0 beginnen.

**VBS** In VBScript bezeichnen Sie den Index eines Elements, indem Sie den Kollektionsnamen gefolgt durch den Index in runden Klammern verwenden.

```
appRef.Documents.Add
appRef.Documents(1).Layers.Add
appRef.Documents(1).Layers(1).TextFrames.Add
```

## Nummerierung von Objekten durch Elemente und Kollektionen

Nachfolgend wird beschrieben, wie Skriptsprachen die automatische Nummerierung durchführen, wenn Sie ein zweites Objekt zu einer Kollektion oder einem Element hinzufügen:

- ▶ AS weist dem neuen Objekt die Zahl 1 zu und nummeriert das vorher vorhandene Objekt um, sodass es nun die Nummer 2 ist. AppleScript-Objektzahlen wechseln zwischen den Objekten, um das zuletzt bearbeitete Objekt anzugeben. Bei längeren Skripten kann dies verwirrend sein. Daher sollten Skriptanfänger Variablen als Objektverweise verwenden und Indizes vermeiden.
- ▶ JS-Kollektionsnummern sind statisch. Sie ändern sich nicht, wenn Sie ein neues Objekt der Kollektion hinzufügen. Die Objektnumerierung in JS gibt die Reihenfolge an, in der Objekte der Kollektion hinzugefügt wurden. Da dem zuerst hinzugefügten Objekt die Nummer 0 zugewiesen wurde, hat das nächste der Kollektion hinzugefügte Objekt die Nummer 1. Wenn Sie ein drittes Objekt hinzufügen, wird ihm die Nummer 2 zugewiesen. Wenn Sie z. B. ein Dokument hinzufügen, enthält dieses Dokument automatisch eine Ebene. Der Index der Ebene ist [0]. Wenn Sie eine Ebene hinzufügen, ist der Index der neuen Ebene [1]. Wenn Sie eine zweite Ebene hinzufügen, ist deren Index [2]. Wenn Sie die Ebene [2] an die untere Position in der Ebenen-Palette ziehen, hat sie immer noch den Index [2].
- ▶ Die VBS-Kollektionsnummern sind ebenfalls statisch und die Nummerierung wird wie für die JS-Kollektionen beschrieben durchgeführt, mit der Ausnahme, dass das erste Objekt in der Kollektion in VBS stets (1) ist.

**TIPP:** In JS- und VBS-Skripten sind Indexnummern hilfreich als Objektverweise. Beispiel: Sie haben mehrere Dateien, in denen die Hintergrundfarbe weiß sein soll. Sie können ein Skript schreiben mit dem Inhalt „Alle Dateien in diesem Ordner öffnen und die Farbe der ersten Ebene in weiß ändern.“ Wenn Sie auf die Ebenen nicht durch Index verweisen können, müssen Sie in das Skript die Namen aller Hintergrundebenen in allen Dateien aufnehmen.

**HINWEIS:** Skripte sind Ordnungsfanatiker. Sie platzieren Objekte auch dann in Elementen oder Kollektionen, wenn nur ein Objekt dieses Typs in der gesamten Kollektion vorhanden ist.

**HINWEIS:** Objekte können mehr als einer Kollektion oder einem Element angehören. Beispiel: In Photoshop gehören `art layer`-Objekte zum `art layers`-Element oder zur `art layers`-Kollektion und `layer set`-Objekte gehören zum `layer sets`-Element oder zur `layer sets`-Kollektion. Sowohl `art layer`-Objekte als auch `layer set`-Objekte gehören zum `layers`-Element oder zur `layers`-Kollektion. In InDesign gehören `rectangle`-Objekte zum `rectangles`-Element bzw. zur `rectangle`-Kollektion und `text frame`-Objekte gehören zum `text frames`-Element bzw. zur `text frames`-Kollektion. Sowohl `rectangle`-Objekte als auch `text frame`-Objekte gehören zum `page items`-Element bzw. zur `page items`-Kollektion, das/die alle Arten von Elementen auf einer Seite wie Ellipsen, Grafiklinien, Polygone, Schaltflächen und andere Elemente enthält.

## Verweise auf das aktuelle oder aktive Objekt

Als Sie das erste Skript ausgeführt und ein neues Dokument erstellt haben, hat sich die Anwendung geöffnet und ein Dokument wurde erstellt. Wenn Sie das Dokument in der Benutzeroberfläche der Anwendung ändern wollten, konnten Sie dazu direkt die Maus, Menüs, den Werkzeugkasten und Paletten verwenden, da das Dokument automatisch ausgewählt wurde.

Dies gilt für alle Objekte, die Sie in einem Skript erstellen. Bis das Skript andere Funktionen ausführt, ist das neue Objekt das aktive Objekt, das geändert werden kann.

Viele übergeordnete Objekte enthalten Eigenschaften, mit denen Sie ganz einfach auf das aktive Objekt verweisen können. (Eigenschaften werden in diesem Handbuch noch ausführlicher dargestellt. Im Moment können Sie die Skriptanweisungen in diesem Abschnitt einfach kopieren und beobachten, wie sie funktionieren – ohne zu wissen, was es genau mit ihnen auf sich hat.)

- ▶ In AS besteht die Eigenschaft, die auf ein aktives Objekt verweist, aus dem Wort `current` und dem Objektnamen. Beispiele:

```
current document
current layer
current channel
current view
```

- ▶ In JS ist der Eigenschaftename ein zusammengesetztes Wort, das `active` mit dem Objektnamen kombiniert, in standardmäßiger Verwendung der JS-Groß-/Kleinschreibung.
  - ▷ Das erste Wort in dem zusammengesetzten Ausdruck ist klein geschrieben.
  - ▷ Das zweite Wort (und alle folgenden Wörter) im zusammengesetzten Ausdruck beginnen mit Großbuchstaben.

Beispiele:

```
activeDocument
activeLayer
activeChannel
activeView
```

- ▶ VBS entspricht genau JS, mit der Ausnahme, dass alle Wörter in den zusammengesetzten Ausdrücken Großbuchstaben am Wortanfang verwenden. Beispiele:

```
ActiveDocument
ActiveLayer
ActiveChannel
ActiveView
```

Die folgenden Skripte erstellen ein Dokument und verwenden anschließend dieses Prinzip, um eine Ebene in diesem neuen Dokument einzurichten.

**AS**

```
tell application "Adobe Illustrator CS6"
  make document
  make layer in current document
end tell
```

**JS**

```
app.documents.add()
app.activeDocument.layers.add()
```

**HINWEIS:** Achten Sie darauf, `activeDocument` ohne ein `s` am Ende einzugeben.

**VBS**

```
Set appRef = CreateObject("Illustrator.Application")
docRef.Documents.Add
appRef.ActiveDocument.Layers.Add
```

**HINWEIS:** Achten Sie darauf, `ActiveDocument` ohne ein `s` am Ende einzugeben.

## Eigenschaften verwenden

Um eine Eigenschaft eines Objekts zu definieren oder zu ändern, sind drei Schritte erforderlich:

1. Benennen des Objekts.
2. Benennen der Eigenschaft.
3. Festlegen des Werts für die Eigenschaft.

Der Wert kann einem der folgenden Datentypen entsprechen:

- ▶ Eine Zeichenfolge in Form eines alphanumerischen Textes, der als Text interpretiert wird. Schließen Sie die Zeichenfolge in Anführungszeichen ein (""). Zeichenfolgen enthalten Werte wie den Namen eines Objekts.
- ▶ Numerische Werte sind Zahlenwerte, die in mathematischen Operationen wie Additionen oder Divisionen verwendet werden. Mathematische Zahlen enthalten die Länge einer Rahmenseite oder den Raum zwischen Absätzen, den Prozentsatz der Deckkraft, die Schriftgröße, die Konturstärke usw.



Manche Werte, die wie Zahlen aussehen, sind in Wirklichkeit Zeichenfolgen. Eine Telefonnummer oder die Sozialversicherungsnummer sind Zahlen. Sie werden aber als Zeichenfolgen formatiert (in Anführungszeichen gesetzt), da die Daten nicht als mathematische Zahlen betrachtet werden.

Innerhalb der numerischen Kategorie gibt es verschiedene Arten von Zahlen:

- ▷ Ganzzahlen sind Zahlen ohne Dezimalstellen
- ▷ Reale, feste, kurze, lange oder doppelte Zahlen sind Zahlen, die Dezimalstellen enthalten können, wie 5,9 oder 1,0.

Hinweis: Im Moment spielen diese Unterschiede keine Rolle; wir werden sie uns jedoch später genauer ansehen.

- ▶ Variablen. Wenn Sie eine Variable als Eigenschaftswert verwenden, wird sie nicht wie eine Zeichenfolge in Anführungszeichen gesetzt.
- ▶ Boolesche Werte, die entweder `true` oder `false` sind.

**HINWEIS:** In vielen Fällen können Boolesche Werte als Ein/Aus-Schalter agieren.

- ▶ Ein konstanter Wert (auch *Aufzählung* genannt), der ein vordefinierter Satz von Werten ist, aus denen Sie eine Auswahl treffen können. Das Verwenden von konstanten Werten für eine Eigenschaft entspricht dem Konzept eines Dropdown-Menüs in einer Adobe-Anwendung. Konstante und ihre Verwendung werden erläutert unter [„Konstantenwerte und Aufzählungen“ auf Seite 22](#).
- ▶ Eine Liste (AS) oder ein Array (JS und VBS).

Bei einigen Eigenschaften sind mehrere Werte erforderlich, wie die Seitenkoordinaten eines Punktes (x- und y-Koordinaten) oder die Grenzen eines Textrahmens oder eines geometrischen Objekts. Mehrere Werte für eine Eigenschaft werden in AS „Liste“ und in JS oder VBS „Array“ genannt. Jede Sprache definiert Formatierungsrichtlinien.

- ▷ Die Liste oder das Array muss auf die folgende Art eingeschlossen werden:

In AS werden Listen in geschweifte Klammern gesetzt: `{ }`

In JS werden Arrays in eckige Klammern gesetzt. `[ ]`

In VBS werden Arrays in runde Klammern gesetzt, nach dem Schlüsselwort `Array: Array()`

- ▷ Werte werden durch Kommata voneinander getrennt (,). Es spielt keine Rolle, ob Sie nach den Kommata Leerzeichen setzen oder nicht.

AS     `{ 3,4,5 }` oder `{ "string1", "string2", "string3" }`

JS     `[ 3,4,5 ]` oder `[ "string1", "string2", "string3" ]`

VBS    `Array(3,4,5)` oder `Array("string1", "string2", "string3")`

- ▷ Eine Liste oder ein Array kann verschachtelte Listen oder Arrays enthalten, wie eine Liste mit Seitenkoordinaten. Beachten Sie in den folgenden Beispielen, dass jedes verschachtelte Array einzeln eingeschlossen ist und dass verschachtelte Arrays durch Kommata voneinander getrennt sind.

```
AS    {{x1, y1}, {x2, y2}, {x3, y3}}
```

```
JS    [[x1, y1], [x2, y2], [x3, y3]]
```

```
VBS    Array(Array(x1, y1), Array(x2, y2), Array(x3, y3))
```

## AS

Um Eigenschaften in AS zu verwenden, geben Sie den Befehl `set`, gefolgt vom Namen der Eigenschaft und anschließend `of`, gefolgt vom Objektverweis ein. Das folgende Skript definiert die Eigenschaft `name` des Objekts `layer`.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc
    set name of myLayer to "My New Layer"
end tell
```

Sie können mehrere Eigenschaften in einer Anweisung festlegen, indem Sie die Eigenschaft `properties` verwenden. Formatieren Sie die Eigenschaften als Array, eingeschlossen in geschweiften Klammern. Trennen Sie innerhalb des Arrays die einzelnen Eigenschaftsnamen/Eigenschaftswertepaare durch einen Doppelpunkt (:). Das folgende Skript verwendet `properties`, um den Namen der Ebene und den Sichtbarkeitsstatus zu definieren.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc
    set properties of myLayer to {name:"My New Layer", visible:false}
end tell
```

**HINWEIS:** Im vorherigen Skript wurde nur der Zeichenfolgewert `Eigene neue Ebene` in Anführungszeichen gesetzt. Der Wert für die Eigenschaft `visible = false` – sieht wie eine Zeichenfolge aus, ist jedoch ein Boolescher Wert. Eine Erläuterung zu den Wertetypen finden Sie in [„Eigenschaften verwenden“ auf Seite 16](#).

Sie können die Eigenschaften eines Objekts in der Anweisung definieren, die das Objekt erstellt, wie in den folgenden Skripten.

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc with properties {name:"My New Layer"}
end tell
```

```
tell application "Adobe Illustrator CS6"
    set myDoc to make document
    set myLayer to make layer in myDoc with properties {name:"My New Layer",
        visible:false}
end tell
```

## JS

Um eine Eigenschaft in JS zu verwenden, benennen Sie das Objekt, für das Sie die Eigenschaft definieren oder ändern möchten, und fügen einen Punkt (.) und den Namen der Eigenschaft ein. Um den Wert festzulegen, setzen Sie ein Gleichheitszeichen (=) nach dem Namen der Eigenschaft und geben den Wert ein.

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
    myLayer.name = "My New Layer"
```

Um mehrere Eigenschaften zu definieren, können Sie mehrere Anweisungen schreiben:

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
    myLayer.name = "My New Layer"
    myLayer.visible = false
```

**HINWEIS:** Im vorherigen Skript wurde nur der Zeichenfolgewart "Eigene neue Ebene" in Anführungszeichen gesetzt. Der Wert für die Eigenschaft `visible = false` – sieht wie eine Zeichenfolge aus, ist jedoch ein Boolescher Wert. Eine Erläuterung zu den Wertetypen finden Sie in [„Eigenschaften verwenden“ auf Seite 16](#).

In JS gibt es ein Kürzel, um mehrere Eigenschaften zu definieren – die Anweisung `with`. Um eine Anweisung `with` zu verwenden, können Sie das Wort `with`, gefolgt vom Objekt, dessen Eigenschaften Sie definieren möchten, verwenden und den Objektverweis in Klammern setzen (`()`). Geben Sie kein Leerzeichen zwischen `with` und der ersten Klammer ein. Geben Sie dann eine öffnende geschweifte Klammer ein (`{`), drücken Sie die **Eingabetaste** und geben Sie einen Eigenschaftsnamen und einen Wert in der folgenden Zeile ein. Um die Anweisung `with` zu schließen, geben Sie eine schließende geschweifte Klammer ein (`}`).

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
    with(myLayer) {
        name = "My New Layer"
        visible = false
    }
```

Wenn Sie die Anweisung `with` verwenden, entfällt die Eingabe des Objektverweises gefolgt durch einen Punkt (in diesem Fall `myLayer.`) für jede Eigenschaft. Wenn Sie die Anweisung `with` verwenden, müssen Sie stets die schließende geschweifte Klammer eingeben.

In JS gibt es außerdem die Eigenschaft `properties`, mit der Sie mehrere Werte in einer Anweisung definieren können. Schließen Sie die gesamte Gruppe von Werten in geschweiften Klammern ein (`{}`). Innerhalb der Klammern können Sie einen Doppelpunkt (`:`) verwenden, um einen Eigenschaftsnamen vom Wert zu trennen. Die Eigenschaftsnamen/Eigenschaftswertepaare können Sie durch Kommata voneinander trennen (`,`).

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
    myLayer.properties = {name:"My New Layer", visible:false}
```

## VBS

Um Eigenschaften in VBS zu verwenden, benennen Sie das Objekt, fügen einen Punkt ein (.) und benennen dann die Eigenschaft. Um den Wert festzulegen, setzen Sie ein Gleichheitszeichen (=) nach dem Namen der Eigenschaft und geben den Wert ein.

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.Documents.Add
Set myLayer = myDoc.Layers.Add
    myLayer.Name = "My First Layer"
```

Sie können nur eine Eigenschaft pro Anweisung definieren. Um mehrere Eigenschaften zu definieren, müssen Sie mehrere Anweisungen schreiben:

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.Documents.Add
Set myLayer = myDoc.Layers.Add
    myLayer.Name = "My First Layer"
    myLayer.Opacity = 65
    myLayer.Visible = false
```

**HINWEIS:** Im vorherigen Skript wurde nur der Zeichenfolgewart "Eigene neue Ebene" in Anführungszeichen gesetzt. Der Wert für die Eigenschaft `visible = false` sieht wie eine Zeichenfolge aus, ist jedoch ein Boolescher Wert. Eine Erläuterung zu den Wertetypen finden Sie in [„Eigenschaften verwenden“ auf Seite 16](#).

## Schreibgeschützte Eigenschaften und Lese-/Schreibeigenschaften

Wenn Sie Eigenschaftswerte definieren, können Sie eine Skriptanweisung mit perfekter Syntax schreiben, deren Anweisungen jedoch zu keinem Ergebnis führen. Dies kann passieren, wenn Sie eine Eigenschaft definieren, die *schreibgeschützt* ist.

Beispiel: Die Eigenschaft `name` des Dokumentobjekts ist in den meisten Adobe-Anwendungen schreibgeschützt. Sie können daher den Namen eines vorhandenen Dokuments nicht durch ein Skript definieren oder ändern (obwohl Sie den Befehl bzw. die Methode `save as` verwenden können; weitere Informationen finden Sie in [„Methoden oder Befehle verwenden“ auf Seite 24](#)). Die Frage ist, welchen Zweck eine Eigenschaft hat, die nicht festgelegt werden kann. Schreibgeschützte Eigenschaften sind wichtige Informationsquellen. Beispiel: Sie möchten den Namen eines Dokuments herausfinden oder wissen, wie viele Dokumente sich in der Kollektion `Documents` befinden.

## Warnfelder zum Anzeigen eines Eigenschaftswerts verwenden

Sie können Warnfelder verwenden, um Daten in einer schreibgeschützten Eigenschaft anzuzeigen. Es handelt sich dabei um ein kleines Dialogfeld, das Daten anzeigt. Sie können mit Warnfeldern auch den Wert einer Eigenschaft anzeigen: Schreiben/Lesen oder schreibgeschützt.

## AS

Um ein Warnfeld in AS anzuzeigen, geben Sie `display dialog` und anschließend den Inhalt des Dialogfelds in Klammern `()` ein. Um herauszufinden, wie viele Objekte sich in einem Element befinden, verwenden Sie den Befehl `count` zusammen mit einem Elementnamen.

**HINWEIS:** Der Elementname ist die Pluralform des Objekts. Beispiel: Das Element des Objekts `document` ist das Objekt `documents`.

Das folgende Skript zeigt ein Warnfeld an, das Ihnen mitteilt, wie viele Dokumente sich im Element `documents` befinden. Es fügt dann ein Dokument hinzu und zeigt eine neue Warnung mit der aktualisierten Nummer an.

```
tell application "Adobe Photoshop CS6"
    display dialog (count documents)
    set myDoc to make document
    display dialog (count documents)
end tell
```

Damit ein Zeichenfolgenwert in einem Warnfeld angezeigt wird, müssen Sie den Zeichenfolgewert in einer Variablen speichern. Das folgende Skript konvertiert den Dokumentennamen in eine Variable mit dem Namen `myName` und zeigt den Wert von `myName` an.

```
tell application "Adobe Photoshop CS6"
    set myDoc to make document
    set myName to name of myDoc
    display dialog myName
end tell
```

## JS

Um ein Warnfeld in JS anzuzeigen, verwenden Sie die Methode `alert()`, indem Sie `alert` und anschließend den Inhalt des Dialogfelds in Klammern eingeben `()`. Geben Sie kein Leerzeichen zwischen `alert` und der ersten Klammer ein. Um herauszufinden, wie viele Objekte sich in einer Kollektion befinden, verwenden Sie die (schreibgeschützte) Eigenschaft `length` eines Kollektionsobjekts. Das folgende Skript zeigt ein Warnfeld an, das Ihnen mitteilt, wie viele Dokumente sich in der Kollektion `documents` befinden. Es fügt dann ein Dokument hinzu und zeigt eine neue Warnung mit der aktualisierten Nummer an.

**HINWEIS:** Der Name des Kollektionsobjekts ist die Pluralform des Objekts. Beispiel: Das Kollektionsobjekt des Objekts `document` ist das Objekt `documents`.

```
alert(app.documents.length)
var myDoc = app.documents.add()
alert(app.documents.length)
```

Das folgende Skript zeigt den Dokumentennamen in einem Warnfeld an.

```
var myDoc = app.documents.add()
alert(myDoc.name)
```

## VBS

Um ein Warnfeld in VBS anzuzeigen, verwenden Sie die Methode `MsgBox`, indem Sie `MsgBox` und anschließend den Inhalt des Dialogfelds in Klammern eingeben `( )`. Geben Sie kein Leerzeichen zwischen `MsgBox` und der ersten Klammer ein. Um herauszufinden, wie viele Objekte sich in einer Kollektion befinden, verwenden Sie die (schreibgeschützte) Eigenschaft `Count` eines Kollektionsobjekts. Das folgende Skript zeigt ein Warnfeld an, das Ihnen mitteilt, wie viele Dokumente sich in der Kollektion `Documents` befinden. Es fügt dann ein Dokument hinzu und zeigt eine neue Warnung mit der aktualisierten Nummer an.

**HINWEIS:** Das Kollektionsobjekt ist die Pluralform des Objekts. Beispiel: Das Kollektionsobjekt des Objekts `Document` ist das Objekt `Documents`.

```
Set appRef = CreateObject("Photoshop.Application")
MsgBox (appRef.Documents.Count)
Set myDoc = appRef.Documents.Add
MsgBox (appRef.Documents.Count)
```

Das folgende Skript zeigt den Dokumentennamen in einem Warnfeld an.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add
MsgBox (myDoc.Name)
```

## Konstantenwerte und Aufzählungen

Einige Eigenschaftswerte sind bereits durch die Anwendung definiert. Beispiel: Bei den meisten Anwendungen kann die Seitenausrichtung entweder Hoch- oder Querformat sein. Die Anwendung akzeptiert nur einen dieser beiden Werte. Sie akzeptiert nicht „vertical“ oder „upright“ oder „horizontal“ oder „on its side“. Um zu gewährleisten, dass das Skript einen akzeptablen Wert für die Ausrichtungseigenschaft einer Seite angibt, wurde die Eigenschaft so geschrieben, dass sie nur einen vordefinierten Wert akzeptiert.

Bei Skripten werden diese vordefinierten Werte *Konstanten* oder *Aufzählungen* genannt.

Das Verwenden einer Konstanten oder einer Aufzählung entspricht dem Verwenden einer Dropdown-Liste in der Benutzeroberfläche der Anwendung.

**HINWEIS:** Um herauszufinden, ob Sie eine Aufzählung für einen Eigenschaftswert verwenden müssen, sehen Sie sich die Eigenschaft in einem der Skriptreferenzen von Adobe an. Weitere Informationen finden Sie in [Kapitel 3, „Eigenschaften von Objekten und Methoden“](#).

## AS

In AS verwenden Sie Konstanten wie andere Eigenschaftsdefinitionen. Setzen Sie die Konstante nicht in Anführungszeichen. Das folgende Skript verwendet den Konstantenwert `dark green`, um die Ebenenfarbe einer neuen Ebene festzulegen.

```
tell application "Adobe Illustrator CS6"
  set myDoc to make document
  set myLayer to make layer in myDoc
  set layer color of myLayer to dark green
end tell
```

**HINWEIS:** Wenn `dark green` ein Zeichenfolgenwert und keine Konstante wäre, würde der Wert in Anführungszeichen gesetzt:

```
set layer color of myLayer to "dark green"
```

## JS

In JS geben Sie den Aufzählungsnamen, einen Punkt (.) sowie den Aufzählungswert ein. Sie müssen die genaue Schreibweise sowie die Großschreibung wie in den Skriptreferenzen von Adobe definiert verwenden. Die Formatierungen unterscheiden sich in den einzelnen Adobe-Anwendungen. Beispiel:

### ► In InDesign:

- ▷ Jede Aufzählung beginnt mit einem Großbuchstaben und alle Wörter im zusammengesetzten Ausdruck beginnen ebenfalls mit einem Großbuchstaben.
- ▷ Der Aufzählungswert beginnt mit einem Kleinbuchstaben.

Im folgenden Beispiel wird die Aufzählung `UIColor` verwendet, um die Ebenenfarbe auf dunkelgrün festzulegen.

```
var myDoc = app.documents.add()
var myLayer = mydoc.layers.add()
    myLayer.layerColor = UIColor.darkGreen
```

### ► In Illustrator:

- ▷ Jede Aufzählung beginnt mit einem Großbuchstaben und alle Wörter im zusammengesetzten Ausdruck beginnen ebenfalls mit einem Großbuchstaben.
- ▷ Manche Aufzählungswerte beginnen mit einem Großbuchstaben und verwenden dann Kleinbuchstaben. Andere verwenden durchgehend Großbuchstaben. Achten Sie darauf, die Werte genau so zu verwenden, wie sie in der Skriptreferenz angegeben sind.

Im folgenden Beispiel wird die Aufzählung `RulerUnits` verwendet, um die Standardeinheit auf Zentimeter festzulegen.

```
var myDoc = app.documents.add()
    myDoc.rulerUnits = RulerUnits.Centimeters
```

Das nächste Skript verwendet die Aufzählung `BlendModes`, deren Werte ausschließlich in Großbuchstaben ausgedrückt werden.

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
    myLayer.blendingMode = BlendModes.COLORBURN
```

### ► In Photoshop:

- ▷ Jede Aufzählung beginnt mit einem Großbuchstaben und alle Wörter im zusammengesetzten Ausdruck beginnen ebenfalls mit einem Großbuchstaben.
- ▷ Aufzählungswerte werden ausschließlich in Großbuchstaben eingegeben.

Im folgenden Beispiel wird die Aufzählung `LayerKind` verwendet, um die Ebene als Textebene zu definieren.

```
var myDoc = app.documents.add()
var myLayer = mydoc.artLayers.add()
```

```
myLayer.kind = LayerKind.TEXT
```

## VBS

In VBS verwenden Sie numerische Werte für Konstanten.

```
Set appRef = CreateObject("Photoshop.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.ArtLayers.Add
    layerRef.Kind = 2
```

## Variablen für Eigenschaftswerte verwenden

Variablen können Eigenschaftswerte enthalten. Sie können mit Variablen ein Skript schnell und präzise aktualisieren. Beispiel: Eine Veröffentlichung enthält Fotos, die alle im Format 7,6 x 12,7 cm vorliegen. Wenn Sie Variablen verwenden, um die Länge und Breite eines Fotos festzulegen und dann die Maße ändern, müssen Sie nur die Werte einer Variablen ändern und nicht die Maße für jedes Foto im Dokument.

Die folgenden Skripte erstellen Variablen, die die Werte für die Länge und Breite des Dokuments enthalten. Die Variablen werden anschließend als Werte in der Anweisung verwendet, die die Höhe und die Breite ändert.

```
AS tell application "Adobe Illustrator CS6"
    set myDoc to make document with properties {height:10, width:7}
    set docHeight to height of myDoc
    set docWidth to width of myDoc
    set myDoc with properties {height:docHeight - 2, width:docWidth - 2}
end tell
```

```
JS var myDoc = app.documents.add(7, 10)
var docHeight = myDoc.height
var docWidth = myDoc.width
    myDoc.resizeCanvas((docHeight - 2), (docWidth - 2))
```

```
VBS Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add(7, 10)
docHeight = myDoc.Height
docWidth = myDoc.Width
myDoc.ResizeCanvas docWidth - 2, docHeight - 2
```

**HINWEIS:** Die Methode `MsgBox` funktioniert nicht, wenn Sie ein Skript aus dem Skriptmenü in einer Adobe-Anwendung öffnen. Um das Meldungsfeld richtig anzuzeigen, doppelklicken Sie auf die Skriptdatei in Windows Explorer®.

## Methoden oder Befehle verwenden

Befehle (in AS) und Methoden (in VBS und JS) sind Anweisungen, die Sie einem Skript hinzufügen, um Aufgaben durchzuführen oder um Ergebnisse zu erzielen. Beispiel: Sie können den Befehl/die Methode `print/print()/PrintOut` verwenden, um ein Dokument zu drucken.

**AS** AS-Befehle werden am Anfang einer Skriptanweisung als imperatives Verb angezeigt. Dem Befehl folgt ein Verweis auf das Objekt, für das der Befehl gelten soll.



Das folgende Skript druckt das aktive Dokument:

```
tell application "Adobe InDesign CS6"
    print current document
end tell
```

**JS** Fügen Sie Methoden am Ende von JS-Anweisungen ein. Sie müssen einen Punkt vor den Methodennamen setzen und dann den Methodennamen in Klammern hinzufügen (`()`).

```
app.activeDocument.print()
```

**VBS** Fügen Sie Methoden am Ende von VBS-Anweisungen ein. Sie müssen vor den Methodennamen einen Punkt eingeben.

```
Set appRef = CreateObject("Photoshop.Application")
appRef.ActiveDocument.PrintOut
```

## Befehls- oder Methodenparameter

Bei manchen Befehlen oder Methoden sind zusätzliche Daten erforderlich, die *Argumente oder Parameter* genannt werden. Befehlen oder Methoden können auch optionale Parameter zugeordnet werden.

### Erforderliche Parameter

In den folgenden Skripten wird der Befehl `merge` verwendet, für den eine Angabe erforderlich ist, welche Ebenen mit der ausgewählten Ebene zusammengeführt werden sollen. Wie Eigenschaften werden Befehlsparameter in geschweiften Klammern eingeschlossen (`{ }`). Setzen Sie jedoch nur den Parameterwert und nicht den Parameternamen in Klammern.

**HINWEIS:** Dieses Skript gilt für InDesign. In Illustrator gibt es keine Operation zum Zusammenführen. Beachten Sie, wenn Sie dieses Skript für Photoshop verwenden, dass eine Ebene in AS `art layer` genannt wird. Ebenen werden in JS `artLayers` bzw. in VBS `ArtLayers` genannt.

#### AS

```
tell application "Adobe InDesign CS6"
    set myDoc to make document

    set myLayer to make layer in myDoc
    set myLayer2 to make layer in myDoc

    merge myLayer2 with {myLayer}
end tell
```

#### JS

Der Methodenparameter wird in Klammern gesetzt und folgt dem Methodennamen.

```
var myDoc = app.documents.add()

var myLayer = myDoc.layers.add()
var myLayer2 = myDoc.layers.add()

myLayer2.merge(myLayer)
```

**VBS**

Beachten Sie, dass der Methodenparameter in Klammern gesetzt ist und dem Methodennamen folgt. Geben Sie nach der ersten Klammer kein Leerzeichen ein.

```
Set appRef = CreateObject("InDesign.Application")
Set myDoc = appRef.Documents.Add

Set myLayer = myDoc.Layers.Add
Set myLayer2 = myDoc.Layers.Add

myLayer2.Merge(myLayer)
```

**Mehrere Parameter**

Wenn Sie mehr als einen Parameter für einen Befehl oder eine Methode definieren, müssen Sie bestimmte Richtlinien beachten.

**AS**

Es gibt zwei Typen von Parametern für AS-Befehle:

- ▶ *Direkte* Parameter, die das direkte Objekt der Aktion definieren, die durch den Befehl ausgeführt wird
- ▶ *Bezeichnete* Parameter sind alle Parameter mit Ausnahme der direkten Parameter.

Direkte Parameter müssen direkt dem Befehl folgen. In der folgenden Anweisung lautet der Befehl `make` und der direkte Parameter `document`.

```
make document
```

Sie können bezeichnete Parameter in jeder beliebigen Reihenfolge einfügen. Das folgende Skript erstellt zwei Ebenen und definiert die Position und den Namen der Ebenen. Beachten Sie, dass in der Anweisung, durch die die Ebenen erstellt werden, die Parameter `location` und `name` in verschiedenen Reihenfolgen vorkommen.

```
tell application "Adobe InDesign CS6"
  set myDoc to make document
  tell myDoc
    set myLayer to make layer at beginning of myDoc with properties {name:"Lay1"}
    set myLayer2 to make layer with properties {name:"Lay2"} at end of myDoc
  end tell
end tell
```

**JS**

In JS müssen Sie die Parameterwerte in der Reihenfolge eingeben, in der sie in den Skriptreferenz-Ressourcen aufgeführt sind, so dass der Skript-Compiler weiß, welcher Wert welchen Parameter definiert.

**HINWEIS:** Informationen über Skriptreferenz-Ressourcen finden Sie in [Kapitel 3, „Eigenschaften von Objekten und Methoden“](#).

Um einen optionalen Parameter zu überspringen, geben Sie den Platzhalter `undefined` ein. Die folgende Anweisung erstellt ein Photoshop CS6-Dokument mit einer Breite von 4000 Pixel, einer Höhe von 5000 Pixel, einer Auflösung von 72, dem Namen „Eigenes Dokument“ und dem Dokumentmodus Bitmap.

```
app.documents.add(4000, 5000, 72, "My Document", NewDocumentMode.BITMAP)
```

Die nächste Anweisung erstellt ein identisches Dokument mit der Ausnahme, dass die Auflösung nicht definiert wird.

```
app.documents.add(4000, 5000, undefined, "My Document", NewDocumentMode.BITMAP)
```

**HINWEIS:** Verwenden Sie den Platzhalter `undefined` nur, um den zu definierenden Parameter zu „erreichen“. Die folgende Anweisung definiert nur die Länge und Breite des Dokuments; Platzhalter werden nicht für die folgenden optionalen Parameter benötigt.

```
app.documents.add(4000, 5000)
```

## VBS

In VBS müssen Sie die Parameterwerte in der Reihenfolge eingeben, in der sie aufgelistet sind, so dass der Skript-Compiler weiß, welcher Wert welchen Parameter definiert.

Um einen optionalen Parameter zu überspringen, geben Sie den Platzhalter `undefined` ein. Die folgende Anweisung erstellt ein Photoshop CS6-Dokument mit einer Breite von 4000 Pixel, einer Höhe von 5000 Pixel, einer Auflösung von 72, dem Namen „Eigenes Dokument“ und dem Dokumentmodus Bitmap.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add(4000, 5000, 72, "My Document", 5)
```

Die nächste Anweisung erstellt ein identisches Dokument mit der Ausnahme, dass die Auflösung nicht definiert wird.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add(400, 500, undefined, "My Document", 5)
```

**HINWEIS:** Verwenden Sie den Platzhalter `undefined` nur, um den zu definierenden Parameter zu „erreichen“. Die folgende Anweisung definiert nur die Länge und Breite des Dokuments; Platzhalter werden nicht für die folgenden optionalen Parameter benötigt.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Documents.Add(4000, 5000)
```

Beim Platzhalter `undefined` spielt die Groß-/Kleinschreibung keine Rolle.

## Tell-Anweisungen (nur AS)

Vermutlich ist es Ihnen aufgefallen, dass AppleScript-Beispiele mit den folgenden Anweisungen beginnen und enden:

```
tell application "Anwendungsname"
end tell
```

Die Anweisung `tell` benennt das Standardobjekt, das alle in der Anweisung enthaltenen Befehle durchführt. Im vorhergehenden Beispiel zielt die Anweisung `tell` auf das Anwendungsobjekt ab. Daher müssen alle in dieser Anweisung enthaltenen Befehle durch das Anwendungsobjekt durchgeführt werden, sofern kein anderes Objekt explizit in einer Skriptanweisung innerhalb der Anweisung `tell` benannt wird.

Das folgende Skript bezeichnet genau die vollständige Container-Hierarchie jedes Objekts, um anzugeben, für welches Objekt der Befehl gilt:

```
tell application "Adobe InDesign CS6"
  set myDoc to make document
  set myLayer to make layer in myDoc
  set myLayer2 to make layer in myDoc
end tell
```

Sie können eine Verknüpfung erstellen, indem Sie das Befehlsziel ändern. Hierzu fügen Sie eine verschachtelte `tell`-Anweisung hinzu. Das folgende Skript führt genau dieselbe Operation wie das vorherige Skript durch. Da das Ziel der verschachtelten `tell`-Anweisung das Objekt „document“ ist, ist kein Verweis auf dieses Objekt in den Anweisungen erforderlich, die die Ebenen erstellen.

```
tell application "Adobe InDesign CS6"
  set myDoc to make document
  tell myDoc
    set myLayer to make layer
    set myLayer2 to make layer
  end tell
end tell
```

Beachten Sie, dass jede `tell`-Anweisung durch ihre eigene Anweisung `end tell` geschlossen werden muss.

Sie können so viele `tell`-Anweisungen verschachteln, wie Sie möchten.

## Hinweise über Variablen

In diesem Abschnitt finden Sie zusätzliche Informationen über das Verwenden von Variablen.

### Variablenwert ändern

Sie können den Wert einer Variablen jederzeit ändern. Hierzu müssen Sie lediglich den Variablennamen, gefolgt vom Zuweisungsoperator (`to` in AS; `=` in JS oder VBS) und den neuen Wert verwenden. Die folgenden Skripte erstellen die Variable `layerRef`, die eine neue Ebene enthält, und erstellen anschließend sofort eine zweite Ebene und weisen sie als neuen Wert von `layerRef` zu.

**AS** Um den Wert einer Variablen in AS zu ändern, verwenden Sie den Befehl `set`.

```
tell application "Adobe Illustrator CS6"
  set docRef to make document
  set layerRef to make layer in myDoc with properties {name:"First Layer"}
  set layerRef to make layer in myDoc with properties {name:"Second Layer"}
end tell
```

**JS** Um den Wert einer Variablen in JS zu ändern, verwenden Sie den Variablennamen, gefolgt von einem Gleichheitszeichen (`=`) und dem neuen Wert. Beginnen Sie die Anweisung für die Neuzuweisung nicht mit `var`; verwenden Sie `var` nur, wenn Sie eine neue Variable erstellen.

```
var docRef = app.documents.add()
var layerRef = myDoc.layers.add()
layerRef.name = "First Layer"
layerRef = myDoc.layers.add()
layerRef.name = "Second Layer"
```

**VBS** Um den Wert einer Variablen in VBS zu ändern, verwenden Sie den Befehl `Set`.

```
Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.Layers.Add
    layerRef.Name = "First Layer"
    layerRef = docRef.Layers.Add
    layerRef.Name = "Second Layer"
```

## Variablen verwenden, um auf vorhandene Objekte zu verweisen

Sie können auch Variablen erstellen, die vorhandene Objekte enthalten.

**AS**

```
tell application "Adobe Photoshop CS6"
    set myDoc to active document
end tell
```

**JS**

```
var myDoc = app.activeDocument
```

**VBS**

```
Set appRef = CreateObject("Illustrator.Application")
Set docRef = appRef.ActiveDocument
```

## Skriptdateien lesbar gestalten

Dieser Abschnitt enthält zwei Optionen, die Ihnen helfen, die Skriptdateien lesbarer zu gestalten:

- ▶ Kommentare
- ▶ Zeilenumbrüche

### Skript kommentieren

Ein Skriptkommentar ist ein Text, der von der Skript-Engine beim Ausführen des Skripts ignoriert wird.

Kommentare sind sehr hilfreich, wenn Sie die Operation oder den Zweck eines Skripts dokumentieren möchten (für Sie selbst oder für eine andere Person). Die meisten Programmierer, auch wenn sie bereits sehr erfahren sind, nehmen sich die Zeit, um Kommentare für fast alle Elemente in einem Skript einzufügen. Kommentare scheinen Ihnen vielleicht beim Schreiben des Skripts nicht wichtig zu sein. Wenn Sie jedoch das Skript nach einem Monat oder einem Jahr öffnen und sich fragen, was Ihr Ziel war und weshalb, sind Kommentare sehr hilfreich.

**AS** Um eine ganze oder einen Teil einer Zeile in AS zu kommentieren, geben Sie zwei Bindestriche (`--`) am Beginn des Kommentars ein. Um mehrere Zeilen zu kommentieren, setzen Sie den Kommentar zwischen (`*` und `*`).

```
tell application "Adobe InDesign CS6"
--This is a single-line comment
print current document --this is a partial-line comment
--the hyphens hide everything to their right from the scripting engine
(* This is a multi-line
    comment, which is completely
    ignored by the scripting engine, no matter how
    many lines it contains.
    The trick is to remember to close the comment.
    If you don't the rest of your script is
    hidden from the scripting engine!*)
end tell
```

**HINWEIS:** Die einzige Aktion des Skripts besteht darin, das aktuelle Dokument zu drucken.

**JS** Um eine ganze oder einen Teil einer Zeile in JS zu kommentieren, geben Sie zwei Schrägstriche (//) am Beginn des Kommentars ein. Um mehrere Zeilen zu kommentieren, setzen Sie den Kommentar zwischen /\* und \*/.

```
//This is a single-line comment
app.activeDocument.print() //this part of the line is also a comment

/* This is a multi-line
comment, which is completely
ignored by the scripting engine, no matter how
many lines it contains.
Don?t forget the closing asterisk and slash
or the rest of your script will be commented out...*/
```

**HINWEIS:** Die einzige Aktion des Skripts besteht darin, das aktive Dokument zu drucken.

**VBS** In VBS geben Sie Rem (für „Remark“ [Anmerkung]) oder ' (ein einzelner gerader Anführungsstrich) am Beginn des Kommentars ein. VBS unterstützt keine Kommentare, die sich über mehr als eine Zeile erstrecken. Um mehrere Zeilen nacheinander zu kommentieren, beginnen Sie jede Zeile mit einem Kommentarformat.

```
'This is a comment.
Set appRef = CreateObject("Photoshop.Application")
Rem This is also a comment.
appRef.ActiveDocument.PrintOut 'This part of the line is a comment.
' This is a multi-line
' comment that requires
' a comment marker at the beginning
' of each line.
Rem This is also a multi-line comment. Generally, multi-line
Rem comments in VBS are easier for you to identify (and read) in your scripts
Rem if they begin with a single straight quote (') rather than if they begin
Rem with Rem, because Rem can look like any other text in the script
' The choice is yours but isn?t this more easily
' identifiable as a comment than the preceding
' four lines were?
```

**HINWEIS:** Die einzige Aktion des Skripts besteht darin, das aktive Dokument zu drucken.

## Fortlaufende lange Zeilen in AppleScript und VBScript

Sowohl in AppleScript als auch in VBScript signalisiert ein Zeilenumbruch am Ende der Zeile das Ende einer Anweisung. Wenn die Skriptzeilen zu lang sind, um in eine Zeile zu passen, können Sie spezielle *Umbruchzeichen* verwenden – Zeichen, die einen Zeilenumbruch bewirken, das Skript jedoch dazu anweisen, die umbrochene Zeile als legitime Anweisung zu interpretieren.

**HINWEIS:** Sie können das Skripteditor-Fenster auch erweitern, um mit der Anweisung in einer Zeile fortzufahren.

**AS** Geben Sie das Zeichen `↵` (**Option+Umschalttaste**) ein, um eine lange Zeile umzubrechen, aber mit der Anweisung fortzufahren.

```
tell application "Adobe InDesign CS6"
  set myDoc to make document
  set myLayer to make layer in myDoc with properties {name:"Eigene erste Ebene"} at the↵
  beginning of myDoc (* ohne dieses Zeilenumbruchszeichen würde AS diese
    Zeile als unvollständige Anweisung interpretieren*)
  (* Beachten Sie, dass Umbruchzeichen in mehrzeiligen Kommentaren wie diesem
    nicht erforderlich sind*)
  set myLayer2 to make layer in myDoc with properties {name:"Eigene weitere Ebene"} "↵
  before myLayer
end tell
```

**VBS** Geben Sie einen Unterstrich ein (`_`), gefolgt von einem Zeilenumbruch, um einen Umbruch für eine lange Zeile durchzuführen, aber mit der Anweisung fortzufahren.

**HINWEIS:** In beiden Sprachen verliert das Umbruchzeichen seine Funktion, wenn es innerhalb einer Zeichenfolge platziert wird (d. h. innerhalb der Anführungszeichen). Wenn sich der Zeilenumbruch innerhalb eines Zeichenfolgenwerts befindet, platzieren Sie das Umbruchzeichen vor der Zeichenfolge und fügen den Zeilenbruch früher ein.

**HINWEIS:** In JavaScript können Anweisungen Zeilenumbrüche enthalten. Umbruchzeichen sind also nicht erforderlich. Der ExtendScript-Interpreter interpretiert jedoch jede Zeile als vollständige Anweisung. Im Allgemeinen sollten daher Rückläufe nur am Ende von Anweisungen eingefügt werden.

## Arrays verwenden

In VBScript und JavaScript entsprechen Arrays in etwa Kollektionen. Im Gegensatz zu Kollektionen werden Arrays jedoch nicht automatisch erstellt.

Sie können sich ein Array als Liste von Werten für eine einzelne Variable vorstellen. Beispiel: das folgende JavaScript-Array listet vier Werte für die Variable `myFiles` auf:

```
var myFiles = new Array ()
  myFiles[0] = "clouds.bmp"
  myFiles[1] = "clouds.gif"
  myFiles[2] = "clouds.jpg"
  myFiles[3] = "clouds.pdf"
```

Beachten Sie, dass jeder Wert nummeriert ist. Um einen Wert in einer Anweisung zu verwenden, müssen Sie die Nummer mit eingeben. Die folgende Anweisung öffnet die Datei `clouds.gif`:

```
open(myFiles[1])
```

Das folgende Beispiel enthält dieselben Anweisungen in VBScript:

```
Dim myFiles (4)
  myFiles(0) = "clouds.bmp"
  myFiles(1) = "clouds.gif"
  myFiles(2) = "clouds.jpg"
  myFiles(3) = "clouds.pdf"
appRef.Open myFiles(1)
```

**HINWEIS:** Während Indizes in VBS-Kollektionen stets mit der Nummerierung bei (1) beginnen, können Sie in den VBS-Skripten festlegen, ob von Ihnen erstellte Arrays mit der Nummer (1) oder (0) beginnen. Wie Sie die Startnummer der Array-Indizes festlegen, finden Sie in Lehrbüchern zu VBScript. Informationen über Kollektionen und Indexnummern finden Sie unter [„Objektkollektionen oder Elemente als Objektverweise“ auf Seite 13](#).

## Objekte erstellen

Ihr erstes Skript hat gezeigt, wie Sie ein Objekt mit dem Befehl `make` (AS), der Methode `add()` (JS) oder der Methode `Add` (VBS) des Kollektionsobjekts des Objekts erstellen. Beispiel:

**AS**

```
tell application "Adobe Photoshop CS6"  
    make document  
end tell
```

**JS**

```
app.documents.add()
```

**VBS**

```
Set appRef = CreateObject("Photoshop.Application")  
appRef.Documents.Add()
```

Manche Objekte haben jedoch weder einen Befehl `make` (AS), noch eine Methode `add()` (JS) oder eine Methode `Add` (VBS). Wie Sie Objekte dieses Typs erstellen, finden Sie im Abschnitt „Erstellen neuer Objekte“ im Kapitel der jeweiligen Skriptsprache im Adobe-Skripthandbuch Ihrer Anwendung.

## Weitere Informationen über Skripte

Sie können nun einfache Skripte erstellen, um grundlegende Aufgaben durchzuführen. Um Ihre Skriptkenntnisse zu erweitern, verwenden Sie die folgenden Ressourcen:

- ▶ [„Erweiterte Skripttechniken“ auf Seite 46](#)
- ▶ Das Adobe-Skripthandbuch für Ihre Anwendung
- ▶ [Kapitel 6, „Bibliographie“](#)



# 3 Eigenschaften von Objekten und Methoden

Adobe stellt Ihnen die folgenden Ressourcen zur Verfügung, mit denen Sie Objekte, Methoden oder Befehle, Eigenschaften, Aufzählungen und Parameter suchen und verwenden können, um effiziente Skripte zu erstellen.

- ▶ Objekt-Funktionsverzeichnisse oder Typbibliotheken In jeder skriptfähigen Adobe-Anwendung steht Ihnen eine Referenzbibliothek oder ein Funktionsverzeichnis innerhalb der Skripteditor-Umgebung zur Verfügung.
- ▶ Die Adobe-Skriptreferenzdokumente (im PDF-Format), die sich auf der Installations-CD befinden. (Skriptreferenzdokumente sind nicht für alle Adobe-Anwendungen verfügbar.)

## Skriptumgebungs-Browser verwenden

In diesem Abschnitt wird beschrieben, wie Sie den Objektkatalog für die Skriptumgebung für jede Skriptsprache verwenden.

### AppleScript-Datenfunktionsverzeichnisse

Die AppleScript-Funktionsverzeichnisse sind über die Skripteditor-Anwendung von Apple verfügbar.

#### AppleScript-Funktionsverzeichnisse anzeigen

**HINWEIS:** Der Standardspeicherort für die Skripteditor-Anwendung lautet Programme > AppleScript > Script Editor.



1. Wählen Sie im Skripteditor die Option „Datei“ > „Verzeichnis öffnen“. Das Dialogfeld „Verzeichnis öffnen“ des Skripteditors wird geöffnet.
2. Wählen Sie die Adobe-Anwendung und „Öffnen“. Der Skripteditor öffnet die Adobe-Anwendung und zeigt das Funktionsverzeichnis der Anwendung an.

#### AppleScript-Funktionsverzeichnisse verwenden

Das AS-Funktionsverzeichnis unterteilt Objekte in Suites. Die Suite-Namen weisen auf den Typ des Objekts hin, den die Suite enthält.

**So zeigen Sie Objekteigenschaften an:**

1. Wählen Sie im oberen linken Bereich des Datenfunktionsverzeichnisses die Suite aus, die das Objekt enthält.
2. Wählen Sie das Objekt im oberen mittleren Bereich aus.

**HINWEIS:** Objekte werden durch ein quadratisches Symbol angezeigt: ; Befehle werden durch ein rundes Symbol angezeigt: .

Die Objektbeschreibung wird im unteren Ansichtsbereich angezeigt. Die Elemente und Eigenschaften des Objekts werden unter der Beschreibung aufgeführt. Jedes Element stellt einen Hyperlink zum Objekttyp des Elements dar.

3. Jede Eigenschaftsliste enthält die folgenden Elemente:

- ▷ Namen der Eigenschaft
- ▷ Datentyp in Klammern.

Wenn es sich bei dem Datentyp um ein Objekt handelt, ist der Datentyp ein Hyperlink zum Objekt.

Wenn der Datentyp eine Aufzählung ist, ist der Datentyp „anything“. Die gültigen Werte werden nach der Eigenschaftsbeschreibung aufgeführt und sind durch Schrägstriche (/) voneinander getrennt; die Anmerkung „Can return“: ist ihnen vorangestellt.

- ▷ Zugriffswert:

Wenn das Objekt schreibgeschützt ist, wird r/o nach dem Datentyp angezeigt.

Wenn für das Objekt der Lese-/Schreibzugriff möglich ist, wird kein Zugriffswert angegeben.

- ▷ Eine Beschreibung der Eigenschaft.

1. Suite auswählen, um die Objekte und Befehle im oberen mittleren Bereich anzuzeigen

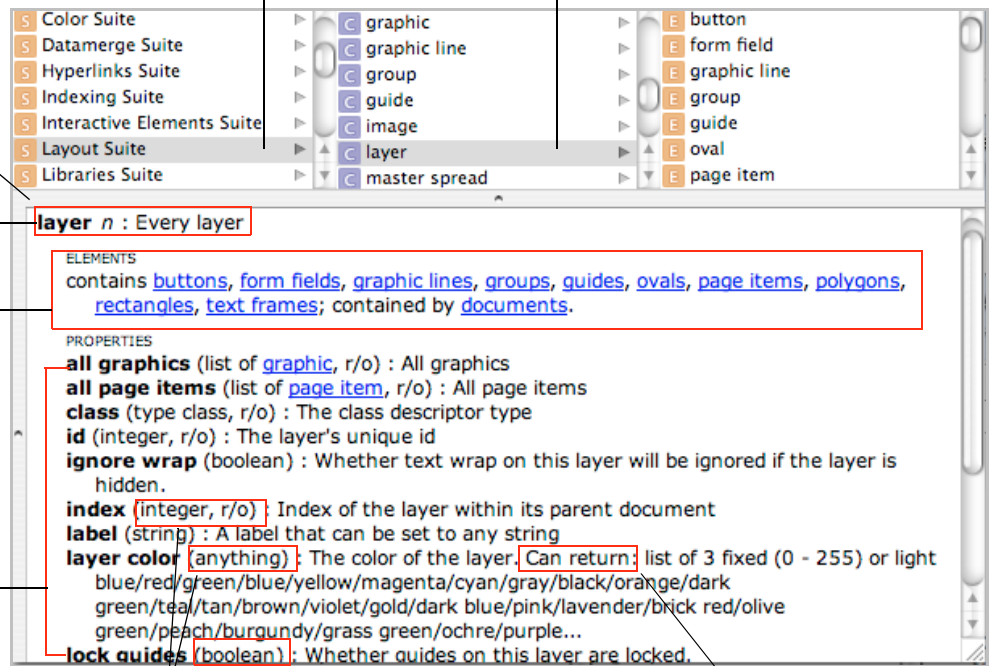
2. Objekt auswählen

3. Objektdaten im unteren Bereich anzeigen

Objektbeschreibung

Verknüpfungen zu den Objektelementen

Eigenschaftsliste



Datentypen und Zugriffswerte werden in Klammern gesetzt, gefolgt vom Eigenschaftsnamen. **Hinweis:** Der Zugriffswert wird nur angezeigt, wenn die Eigenschaft schreibgeschützt ist

Aufgezählten Werten wird „Can return:“ vorangestellt.

## Befehle und Befehlsparameter anzeigen



**HINWEIS:** Das Datenfunktionsverzeichnis listet die Objekte auf, die Sie mit einem Befehl verwenden können. Es führt jedoch nicht die Befehle auf, die Sie mit einem Objekt verwenden können. Wie Sie eine Liste der Befehle anzeigen, die Sie mit einem Objekt verwenden können, finden Sie in der AppleScript-Skriptreferenz für Ihre Anwendung. Weitere Informationen finden Sie in [„Adobe-Skriptreferenzen verwenden“ auf Seite 41](#).

So zeigen Sie Befehle im Datenfunktionsverzeichnis an:

1. Wählen Sie im oberen linken Bereich des Datenfunktionsverzeichnisses die Suite aus, die den Befehl enthält.

Im oberen mittleren Bereich werden die Befehle und Objekte aufgeführt, die in der Suite enthalten sind.

2. Wählen Sie den Befehl im oberen mittleren Bereich aus.

**HINWEIS:** Befehle werden durch ein rundes Symbol angezeigt: ; Objekte werden durch ein quadratisches Symbol angezeigt: .

Die Befehlsbeschreibung wird im unteren Ansichtsbereich angezeigt.

- ▷ Unter der Beschreibung sind die Objekte aufgeführt, die Sie mit dem Befehl verwenden können.
- ▷ Unterhalb der Liste der unterstützten Objekte sind die Parameter aufgeführt.

Optionale Parameter werden in eckige Klammern gesetzt ( [ ] ).

Wenn der Parametername nicht in eckige Klammern gesetzt ist, handelt es sich um erforderliche Parameter.

- ▷ Jedem Parametername folgt der Datentyp.

Wenn es sich bei dem Datentyp um ein Objekt handelt, ist der Datentyp ein Hyperlink zum Objekt.

Wenn es sich bei dem Datentyp um eine Aufzählung handelt, geht den gültigen Werten die Anmerkung „Can accept:“ voraus. Sie werden aufgelistet und durch Schrägstriche voneinander getrennt (/).

1. Suite auswählen, um die Befehle und die Objekte der Suite im oberen mittleren Bereich anzuzeigen

2. Befehl auswählen

3. Befehlsdaten im unteren Bereich anzeigen:

Befehlsbeschreibung

Liste der Objekte, die den Befehl verwenden

Parameter mit Datentypen und Beschreibungen

Optionale Parameter stehen zwischen eckigen Klammern ([ ])

**Hinweis:** Wenn es sich beim Parameterwert um eine Aufzählung handelt, wird den aufgezählten Werten „Can accept:“ vorangestellt

## JavaScript-Objektmodell-Viewer

Sie können das ExtendScript Tools Kit (ESTK) verwenden, das mit den Adobe-Anwendungen installiert wird, um die JavaScript-Objekte und Methoden anzuzeigen, die für die Adobe-Anwendung verfügbar sind.

Weitere Informationen zum Anzeigen und Verwenden des JavaScript-Objektmodell-Viewers für Adobe-Anwendungen finden Sie im *JavaScript Tools Guide*.

## VBScript-Typbibliotheken

Sie können den Visual Basic-Editor in allen Microsoft Office-Anwendungen verwenden, um die VBScript-Objekte und -Methoden für Ihre Adobe-Anwendung anzuzeigen.

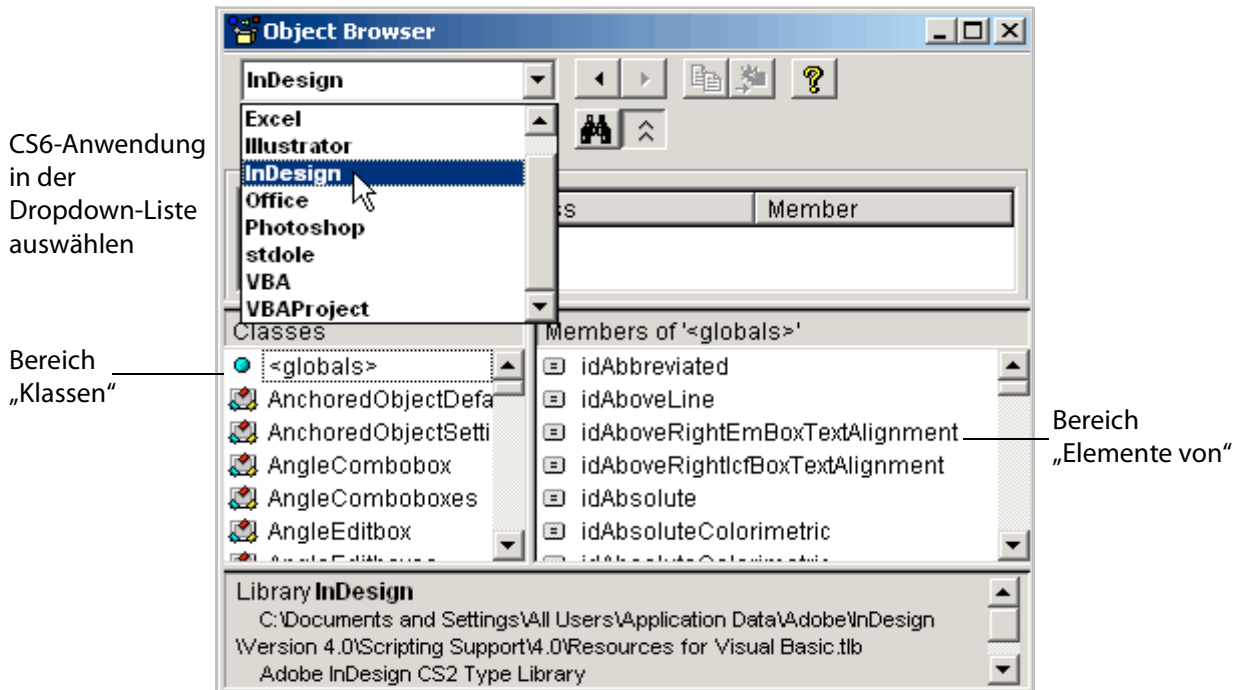
**HINWEIS:** Wenn Sie einen anderen Editor verwenden, finden Sie weitere Einzelheiten zum Anzeigen der Typbibliotheken im Hilfesystem des Editors.

### **VBScript-Typbibliotheken**

So zeigen Sie die VBS-Objektbibliothek an:



1. Wählen Sie in Microsoft Office-Anwendungen „Extras“ > „Makro“ > „Visual Basic-Editor“.
2. Wählen Sie im Visual Basic-Editor „Extras“ > „Verweise“.
3. Wählen Sie im Dialogfeld „Verweise – Projekt“ in der Liste „Verfügbare Verweise:“ die Creative Suite-Anwendung aus und klicken Sie auf „OK“.

4. Wählen Sie im Visual Basic-Editor „Ansicht“ > „Objektkatalog“.
5. Wählen Sie die Adobe-Anwendung in der Dropdown-Liste in der oberen linken Ecke des Fensters „Objektkatalog“ aus.





## VBScript-Typbibliotheken verwenden

Die VBS-Objektypbibliothek zeigt Objekte und Konstanten im Bereich Klassen auf der linken Seite des Fensters Objektkatalog an. Im Bereich Klassen:

- ▶ werden Objekte durch das folgende Symbol gekennzeichnet: 
- ▶ werden Konstante durch das folgende Symbol gekennzeichnet: 

Um die Eigenschaften und die Methode eines Objekts anzuzeigen, wählen Sie den Objekttyp im Bereich „Klassen“ aus. Die Eigenschaften und Methoden werden im Bereich Elemente von auf der rechten Seite des Bereichs Klassen aufgeführt.

- ▶ Eigenschaften werden durch das folgende Symbol gekennzeichnet: 
- ▶ Methoden werden durch das folgende Symbol gekennzeichnet: 

### Eigenschaftenlisten im Objektkatalog

Wenn Sie eine Eigenschaft im Bereich „Elemente von“ auswählen, werden die Eigenschaftsinformationen im Informationsbereich unten im Fenster „Objektkatalog“ wie folgt angezeigt:

- ▶ Jedem Eigenschaftsnamen folgt der Datentyp.
  - ▷ Wenn es sich bei dem Datentyp um eine Konstante handelt, wird die Konstante als Hyperlink zum Konstantenwert angezeigt. Konstantennamen beginnen mit einem Präfix, das dem abgekürzten Namen der Adobe-Anwendung entspricht. Beispiel:
 

Das Präfix `Ps` wird für Aufzählungen in Photoshop CS6 verwendet.  
Beispiele: `PsColorProfileType`, `PsBitsPerChannelType`

Das Präfix `id` wird für Aufzählungen in InDesign CS6 verwendet.  
Beispiele: `idRenderingIntent`, `idPageOrientation`

Das Präfix `Ai` wird für Aufzählungen in Adobe Illustrator CS6 verwendet.)  
Beispiele: `AiCropOptions`, `AiBlendModes`
  - ▷ Wenn es sich bei dem Datentyp um ein Objekt handelt, ist der Objektname ein Hyperlink zum Objekttyp.
- ▶ Der Zugriffswert wird nur angezeigt, wenn die Eigenschaft schreibgeschützt ist. Wenn für die Eigenschaft der Lese-/Schreibzugriff möglich ist, wird kein Zugriffswert angegeben.

Datentyp wird neben dem Eigenschaftsnamen angezeigt.

Zugriffswert wird nur aufgeführt, wenn es sich um einen schreibgeschützten Zugriff handelt.

Eigenschaftsbeschreibung wird unten im Informationsbereich angezeigt.

1. Eigenschaft im Bereich „Elemente von“ auswählen.

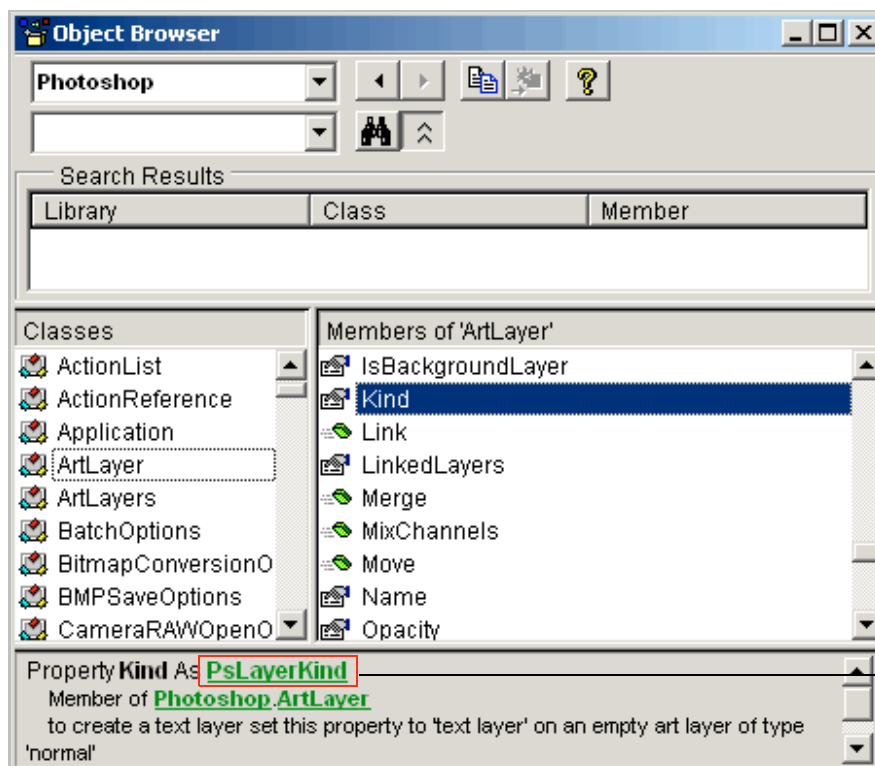
### Numerischen Wert einer Aufzählung suchen

In VBS verwenden Sie den numerischen Wert einer Aufzählung als Eigenschaftswert. Beispiel: Im vorliegenden Skript wird der Ebenentyp, dargestellt durch die Eigenschaft `Kind` in der letzten Zeile des Skripts, durch den numerischen Wert `2` definiert, der den Konstantenwert `TextLayer` darstellt.

```
Set appRef = CreateObject("Photoshop.Application")
Set docRef = appRef.Documents.Add
Set layerRef = docRef.ArtLayers.Add
    layerRef.Kind = 2 'PsTextLayer
```

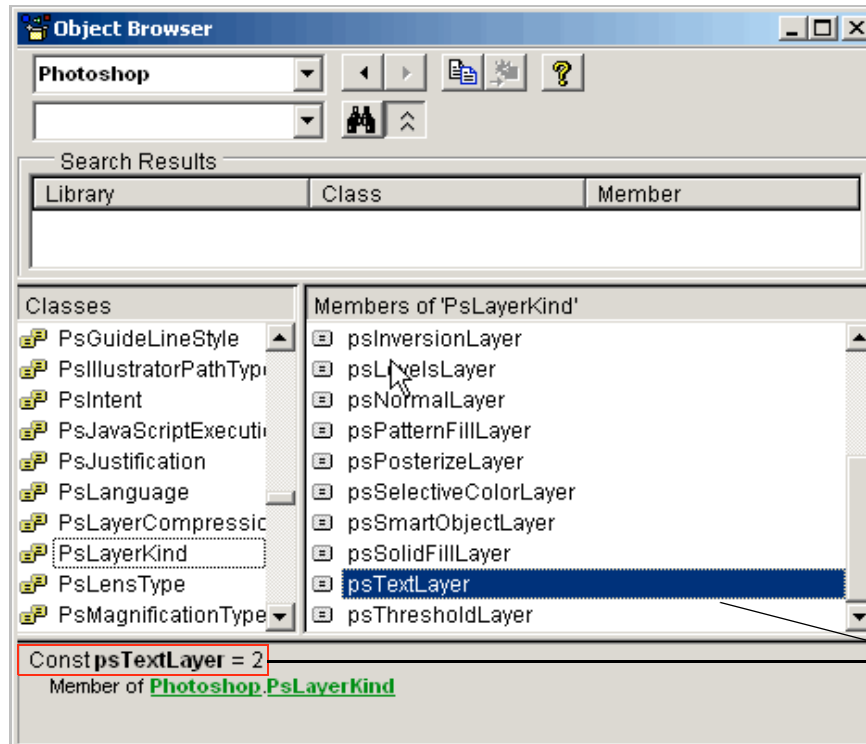
So finden Sie einen numerischen Wert einer Aufzählung:

1. Klicken Sie auf den Link zu den Aufzählungsinformationen.



Klicken Sie auf den Link zu den Aufzählungsinformationen.

2. Klicken Sie auf den Aufzählungswert, um den numerischen Wert im unteren Bereich anzuzeigen.



Auf den Aufzählungswert im rechten Bereich klicken, um den numerischen Wert im unteren Bereich anzuzeigen.

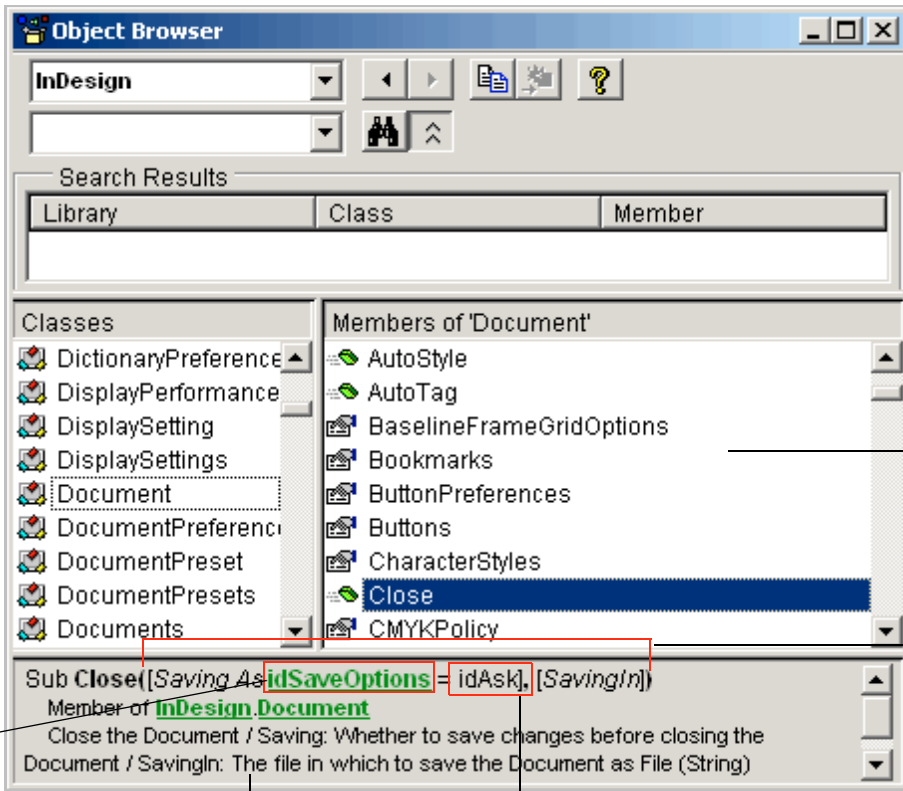
### Methodenaufstellungen

Wenn Sie eine Methode im Bereich „Elemente von“ auswählen, werden die Methodeninformationen im Informationsbereich unten im Fenster „Objektkatalog“ wie folgt angezeigt:

- ▶ Dem Methodennamen folgen die Parameter.
  - ▷ Optionale Parameter stehen zwischen eckigen Klammern ([ ]).
  - ▷ Wenn der Parametername in eckige Klammern gesetzt ist, handelt es sich um einen erforderlichen Parameter.
- ▶ Jedem Parametername folgt der Datentyp.
  - ▷ Wenn es sich bei dem Datentyp um ein Objekt handelt, ist der Datentyp ein Hyperlink zum Objekt.
  - ▷ Wenn es sich bei dem Datentyp um eine Aufzählung handelt, beginnt der Name der Aufzählung mit den Anfangsbuchstaben der Anwendung und ist ein Hyperlink zu den Aufzählungsdaten.
  - ▷ Wenn für einen Parameter ein Standardwert existiert, wird der Wert nach dem Datentyp nach einem Gleichheitszeichen (=) aufgeführt.

**HINWEIS:** Der Standardwert wird verwendet, wenn Sie keinen Wert für den Parameter definieren. Nur optionale Parameter haben Standardwerte.





Der Datentyp wird nach dem Methodennamen aufgeführt. Wenn es sich bei dem Datentyp um eine Aufzählung handelt, beginnt der Aufzählungsname mit den Anfangsbuchstaben der Anwendung und ist ein Link zu den Aufzählungsinformationen

Die Methodenbeschreibung wird unten im Informationsbereich angezeigt.

Wenn ein Standardwert vorhanden ist, folgt er nach einem Gleichheitszeichen (=).  
**Hinweis:** Jeder Datentyp kann einen Standardwert haben.

1. Methode im Bereich „Elemente von“ auswählen.  
 Die Parameter werden in Klammern nach dem Methodennamen aufgeführt, wobei optionale Parameter in eckige Klammern gesetzt werden ([ ]).

## Adobe-Skriptreferenzen verwenden

Adobe stellt für viele Anwendungen Skriptreferenzen zur Verfügung. Die Referenzen befinden sich auf der Installations-CD.

In den Skriptreferenzen wird jede Sprache in einem eigenen Kapitel dokumentiert. Innerhalb der einzelnen Kapitel sind die Objekte alphabetisch aufgeführt. Für jedes Objekt werden die folgenden Tabellen angegeben:

- ▶ Elemente (nur AS)
- ▶ Eigenschaften
- ▶ Methoden, Befehle oder Funktionen

Zusätzlich enthalten die meisten Objektabschnitte Skriptbeispiele, wobei das Objekt und seine Eigenschaften und Methoden oder Befehle verwendet werden. Sie können jedes Beispielskript als Beispiel oder Ausgangspunkt für Ihr Skript verwenden, in dem Sie Eigenschaften oder Methoden ändern können.

## Mit der Elementtabelle eines Objekts arbeiten (nur AS)

Elemente sind die Objektkollektionen in einem Objekt. Wenn ein Objekt Elemente enthält, zeigt eine Tabelle die verschiedenen Arten an, auf die Sie auf Elemente verweisen können. Was die Elementtabelle angeht, ist für Skriptanfänger vor allen Dingen die Spalte Name oder Element wichtig, die angibt, welche Objekte sich unter dem Objekt in der Container-Hierarchie befinden. Beispielsweise stammt die folgende Elementetabelle aus einem `document`-Objekt in InDesign.

Name	Verweis durch
character style	Index, Name, Bereich, relativ, Bestehen eines Tests, ID
layer	Index, Name, Bereich, relativ, Bestehen eines Tests, ID
story	Index, Name, Bereich, relativ, Bestehen eines Tests, ID

Sie können dieser Tabelle entnehmen, dass Sie in Dokumenten, die Sie für diese Anwendung erstellen, Objekte vom Typ `character style`, `layer` und `story` erstellen können.

Beispiel:

```
tell application "Adobe InDesign CS6"
  set myDoc to make document
  set myCharStyle to make character style in myDoc with properties {name:"Bold"}
  set myLayer to make layer in myDoc
  set myStory to make story in myDoc
end tell
```

Die folgende Skriptanweisung würde eine Fehlermeldung generieren, da `stroke style` kein Element des Objekts `document` der Anwendung ist.

```
tell application "Adobe InDesign CS6"
  set myDoc to make document
  set myStrokeStyle to make stroke style in myDoc with properties {name:"Erratic"}
end tell
```

## Mit der Eigenschaftstabelle eines Objekts arbeiten

Die Eigenschaftstabelle für ein Objekt führt die folgenden Elemente auf:

- ▶ Eigenschaften, die Sie zusammen mit dem Objekt verwenden können
- ▶ Wertetyp für jede Eigenschaft

Wenn der Wertetyp eine Konstante oder eine Aufzählung ist, wird der Wert entweder als Liste von gültigen Zeichen oder als Hypertext-Link zur Konstantenauflistung dargestellt.

Wenn der Wertetyp ein anderes Objekt ist, wird der Wert als Hypertext-Link zur Objektauflistung dargestellt.

- ▶ Eingabestatus der Eigenschaft: *Schreibgeschützt* oder *Lesen/Schreiben*

- ▶ Eine Beschreibung, die folgende Elemente enthält:
  - ▷ Eine Erläuterung der Definition oder der Aktion einer Eigenschaft
  - ▷ Bereiche für gültige Werte
  - ▷ Abhängigkeiten von anderen Eigenschaften

Das folgende Beispiel für eine Eigenschaftstabelle eines Objekts `art layer` in Photoshop enthält Beispiele der einzelnen Datentypen.

Eigenschaft	Wertetyp	Eigenschaften
<code>bounds</code>	Array von 4 Zahlen	Schreibgeschützt. Ein Array von Koordinaten, das das Begrenzungsrechteck der Ebene im Format [y1, x1, y2, x2] beschreibt.
<code>kind</code>	<code>LayerKind</code>	Schreibgeschützt. Die Art der Ebene.
<code>name</code>	Zeichenfolge	Lesen/Schreiben. Der Name der Ebene.
<code>opacity</code>	Zahl (zweistellig)	Lesen/Schreiben. Die Deckkraft als Prozentsatz. (Bereich: 0,0 bis 100,0)
<code>textItem</code>	<code>TextItem</code> -Objekt	Schreibgeschützt. Das Textelement, das mit der Ebene verknüpft ist. <b>HINWEIS:</b> Nur gültig, wenn <code>kind = LayerKind.TEXT</code> . Siehe <code>kind</code> .
<code>visible</code>	Boolesch	Lesen/Schreiben. Wenn <code>true</code> , ist die Ebene sichtbar.

Beispiel:

**AS**

```
tell application "Adobe Photoshop CS6"
  set myDoc to make document
  set myLayer to make art layer in myDoc
  set properties of myLayer to {kind:text layer, name:"Captions", opacity:45.5, "
    visible:true}
  set contents of text object in myLayer to "Photo Captions"
end tell
```

**HINWEIS:** Die Begrenzungen der Ebene können Sie nicht definieren, weil die Eigenschaft `bounds` schreibgeschützt ist.

**JS**

```
var myDoc = app.documents.add()
var myLayer = myDoc.artLayers.add()
alert(myLayer.bounds) // can't set the property because it is read-only
myLayer.kind = LayerKind.TEXT
myLayer.name = "Captions"
myLayer.opacity = 45.5 // can use a decimal point because the type is not integer
myLayer.textItem.contents = "Day 1: John goes to school"
//see the properties table for the textItem object to find the contents property
myLayer.visible = true
```

```

VBS      Set appRef = CreateObject("Photoshop.Application")
            Set docRef = appRef.Documents.Add
            Set layerRef = docRef.Layers.Add
            MsgBox(layerRef.Bounds) ' can?t set the property because it is read-only
            layerRef.Kind = 2
            layerRef.Name = "Captions"
            layerRef.Opacity = 45.5 // can use a decimal point because the type is not integer
            layerRef.TextItem.Contents = "Day 1: John goes to school"
            //see the Properties table for the TextItem object to find the Contents property
            layerRef.Visible = true

```

**HINWEIS:** In JS und VBS befinden sich Kollektionsobjekte in den Eigenschaften des enthaltenden Objekts. Um die Container-Hierarchie eines Objekts zu bestimmen, müssen Sie das oder die Objekte lokalisieren, die das Kollektionsobjekt des Objekts (d. h. die Pluralform des Objekts) als Eigenschaft verwenden.

Beispiel: `documents.layers` oder `layers.textFrames`.

## Mit der Methodentabelle von Objekten arbeiten

Die Tabelle Methods eines Objekts führt die folgenden Elemente auf:

- ▶ Methoden, die Sie zusammen mit dem Objekt verwenden können
- ▶ Parameter für jede Methode
  - ▷ Wenn ein Parametertyp eine Konstante oder ein anderes Objekt ist, wird der Wert als Hypertext-Link zur Konstanten oder zur Objektauflistung dargestellt. Im folgenden Beispiel der Tabelle „Methods“ sind die Parametertypen `NewDocumentMode` und `DocumentFill` Konstanten.
  - ▷ Parameter können erforderlich oder optional sein. Optionale Parameter stehen zwischen eckigen Klammern ([ ]).
- ▶ Rückgabewerttyp(en) sind die Produkte der Methode
 

Wenn eine Rückgabe eine Konstante oder ein anderes Objekt ist, wird der Wert als Hypertext-Link auf die Konstante oder die Objektauflistung dargestellt. Im folgenden Beispiel der Tabelle „Methods“ ist der Rückgabewert `Document` ein Objekt.
- ▶ Eine Beschreibung, die die Aktion der Methode definiert

Das folgende Beispiel einer Tabelle Methods führt die Parameter für die Methode `add` für ein Photoshop CS6-Dokument auf.

Methodenname	Parametertyp	Rückgabe	Aktion
<code>add</code>		<code>Document</code>	Fügt ein Dokumentobjekt hinzu.
[width]	<code>UnitValue</code>		
[, height]	<code>UnitValue</code>		( <code>pixelAspectRatio</code>
[, resolution])	Zahl (zweistellig)		Bereich: 0,10 bis 10,00)
[, name]	Zeichenfolge		
[, mode])	<code>NewDocumentMode</code>		
[, initialFill]	<code>DocumentFill</code>		
[, pixelAspectRatio])	Zahl (zweistellig)		

In der vorhergehenden Tabelle:

- ▶ Alle Parameter sind optional, wie durch die eckigen Klammern angegeben.
- ▶ Die Standardwerte für die Parameter `width` und `height` sind die aktuellen Linealeinheiten. Daher ist der Datentyp als `UnitValue` aufgeführt. Wenn die aktuelle Einheit des vertikalen Lineals Zoll und die Einheit des horizontalen Lineals Zentimeter ist, erstellt die folgende Anweisung ein Dokument, das 5 Zoll breit und 7 Zentimeter lang ist:

```
AS:    make document with properties {width:5, height:7}
```

```
JS:    app.documents.add(5, 7)
```

```
VBS:   appRef.Documents.Add(5, 7)
```

- ▶ `mode` und `initialFill` haben konstante Werte.

Die folgenden Skriptanweisungen definieren Werte für jeden Parameter, der in der Beispielstabelle „Methods“ aufgeführt ist.

```
AS    make document with properties {width:5, height:7, resolution:72, " name:"Diary",  
mode:bitmap, initial fill:transparent, pixel aspect ratio: 4.7}
```

```
JS    app.documents.add(5, 7, 72, "Diary", NewDocumentMode.BITMAP,  
DocumentFill.TRANSPARENT, 4.7)
```

```
VBS   appRef.Documents.Add(5, 7, 72, "Diary", 5, 3, 4.7 )
```

# 4 Erweiterte Skripttechniken

Die meisten Skripte verlaufen nicht geradlinig von Anfang bis Ende. Häufig finden sich in Skripten verschiedene Pfade, abhängig von den aus dem aktuellen Dokument erfassten Daten, oder es werden Befehle mehrere Male wiederholt. *Kontrollstrukturen* sind Funktionen der Skriptsprache, mit deren Skriptaktionen durchgeführt werden.

## Konditionalanweisungen

### if-Anweisungen

Eine mündliche Anweisung an die Adobe-Anwendung könnte lauten: „Wenn das Dokument nur eine einzelne Ebene hat, soll eine weitere Ebene erstellt werden.“ Dies ist ein Beispiel für eine *Konditionalanweisung*. Durch Konditionalanweisungen werden Entscheidungen getroffen. Skripte können mit ihnen Bedingungen bewerten, wie z. B. die Anzahl der Ebenen, und dann abhängig vom Ergebnis reagieren. Wenn die Bedingung erfüllt ist, führt das Skript die in der `if`-Anweisung enthaltene Aktion durch. Wenn die Bedingung nicht erfüllt ist, führt das Skript die in der `if`-Anweisung enthaltene Aktion nicht durch.

Alle der folgenden Skripte öffnen ein Dokument und prüfen, ob das Dokument eine einzelne Ebene enthält. Wenn nur eine Ebene vorhanden ist, fügt das Skript eine Ebene hinzu und legt die Fülldeckkraft für die neue Ebene auf 65 % fest.

**AS** Eine `if`-Anweisung in AS beginnt mit dem Wort `if`, gefolgt vom Vergleichsausdruck in Klammern, gefolgt von dem Wort `then`. Sie müssen die `if`-Anweisung mit `end if` schließen.

```
tell application "Adobe Photoshop CS6"
  --modify the hard-drive name at the beginning of the filepath, if needed
  set myFilepath to alias "c:Applications:Adobe Photoshop CS6:Samples:Ducky.tif"
  open myFilepath
  set myDoc to current document
  tell myDoc
    if (art layer count = 1) then
      set myLayer to make art layer with properties {fill opacity:65}
    end if
  end tell
end tell
```

**HINWEIS:** AS verwendet ein einzelnes Gleichheitszeichen (=), um Ergebnisse zu vergleichen.

Schließen Sie nun `Ducky.tif` und führen Sie das Skript erneut aus, wobei Sie `if` auf Folgendes ändern:

```
if (art layer count < 1) then
```

**JS** Eine `if`-Anweisung in JS beginnt mit dem Wort `if`, gefolgt vom Vergleichsausdruck in Klammern. Setzen Sie die Aktion in der `if`-Anweisung in geschweifte Klammern (`{}`).

```
var myDoc = app.open(File("/c/Program Files/Adobe/Adobe Photoshop
CS6/Samples/Ducky.tif"));
if(myDoc.artLayers.length == 1){
  var myLayer = myDoc.artLayers.add()
  myLayer.fillOpacity = 65
}
```

**HINWEIS:** JavaScript verwendet ein doppeltes Gleichheitszeichen (==) zum Vergleichen von Werten im Gegensatz zum einfachen Gleichheitszeichen (=), das zum Zuweisen von Werten zu Eigenschaften oder Variablen verwendet wird.

Schließen Sie nun `Ducky.tif` und führen Sie das Skript erneut aus, wobei Sie `if` auf Folgendes ändern:

```
if (myDoc.artLayers.length < 1) {
```

## VBS

Eine `if`-Anweisung in VBS beginnt mit dem Wort `IF`, gefolgt vom Vergleichsausdruck in Klammern, gefolgt von dem Wort `Then`. Sie müssen die `if`-Anweisung mit `end if` schließen.

```
Set appRef = CreateObject("Photoshop.Application")
Set myDoc = appRef.Open("/c/Program Files/Adobe/Adobe Photoshop
CS6/Samples/Ducky.tif")
  If myDoc.ArtLayers.Count = 1 Then
    Set myLayer = myDoc.ArtLayers.Add
    myLayer.FillOpacity = 65
  End If
```

**HINWEIS:** VBS verwendet ein einfaches Gleichheitszeichen zum Vergleichen und Zuweisen von Werten.

Schließen Sie nun `Ducky.tif` und führen Sie das Skript erneut aus, wobei Sie `if` auf Folgendes ändern:

```
If myDoc.ArtLayers.Count < 1 Then
```

## if else -Anweisungen

Manchmal müssen Sie eine kompliziertere Anforderung formulieren, wie: „Wenn das Dokument über eine Ebene verfügt, soll die Fülldeckkraft der Ebene auf 50 % festgelegt werden. Wenn das Dokument über zwei oder mehr Ebenen verfügt, soll die Fülldeckkraft der aktiven Ebene auf 65 % festgelegt werden.“ In einer solchen Situation ist die Anweisung `if else` angebracht.

## AS

```
tell application "Adobe Photoshop CS6"
  --modify the hard-drive name at the beginning of the filepath, if needed
  set myFilepath to alias "c:Applications:Adobe Photoshop CS6:Samples:Ducky.tif"
  open myFilepath
  set myDoc to current document
  tell myDoc
    if (count of art layers < 2) then
      set fill opacity of current layer to 50
    else
      set fill opacity of current layer to 65
    end if
  end tell
end tell
```

## JS

```
var myDoc = app.open(File("/c/Program Files/Adobe/Adobe Photoshop
CS6/Samples/Ducky.tif"));
if (myDoc.artLayers.length < 2) {
  myDoc.activeLayer.fillOpacity = 50
}
else {
  myDoc.activeLayer.fillOpacity = 65
}
```

```

VBS      Set appRef = CreateObject("Photoshop.Application")
            Set myDoc = appRef.Open("/c/Program Files/Adobe/Adobe Photoshop
            CS6/Samples/Ducky1.tif")
            If myDoc.ArtLayers.Count < 2 Then
                myDoc.ActiveLayer.FillOpacity = 50
            Else
                myDoc.ActiveLayer.FillOpacity = 65
            End If

```

## Schleifen

Sie können durch ein Skript alle Objekte eines bestimmten Typs suchen und ändern. Beispiel: Ein Dokument enthält sichtbare und nicht sichtbare Ebenen und Sie möchten alle Ebenen sichtbar machen. Sie möchten dieses Skript für mehrere Dokumente anwenden; die Dokumente verfügen jedoch über eine verschiedene Anzahl von Ebenen.

In dieser Situation ist die Anweisung `repeat` (AS) oder eine Schleife (JS und VBS) praktisch. Eine Schleife „durchläuft“ eine Kollektion von Objekten und führt eine Aktion für jedes Objekt aus.

Um Skripte in diesem Abschnitt zu verwenden, öffnen Sie die Adobe-Anwendung und erstellen ein Dokument, das über mindestens neun Ebenen verfügt. Ein Teil der Ebenen soll sichtbar und ein Teil ausgeblendet werden. Speichern Sie das Dokument und führen Sie das Skript aus. Ersetzen Sie dabei den Namen der Anwendung und den `layer`-Objektnamen im DOM der Anwendung.

Das Grundprinzip dieser Schleifen ist, dass das Skript die erste Ebene im Element oder in der Kollektion kennzeichnet und die Sichtbarkeit der Ebene auf `true` festlegt und anschließend die nächste Ebene kennzeichnet und die Aktion wiederholt und anschließend die nächste Ebene kennzeichnet, bis jede Ebene erfasst wurde.

```

AS      tell application "Adobe Illustrator CS6"
            set myDoc to current document
            tell myDoc
                set myLayerCount to (count layers)
                set myCounter to 1
                repeat while myCounter <= (myLayerCount + 1)
                    set myLayer to layer myCounter
                    set myLayer with properties {visible:true}
                    --the next statement increments the counter to get the next layer
                    set myCounter to myCounter + 1
                end repeat
            end tell
        end tell

```

Dieses Skript verwendet zwei Variablen, `myLayerCount` und `myCounter`, um eine Ebene zu kennzeichnen und die Ebenen zu nummerieren, bis alle Ebenen im Dokument gekennzeichnet sind.

```

JS      var myDoc = app.activeDocument
            var myLayerCount = myDoc.layers.length
            for(var myCounter = 0; myCounter < myLayerCount; myCounter++)
                {var myLayer = myDoc.layers[myCounter]
                myLayer.visible = true}

```



Dieses Skript verwendet eine Schleife `for`, eine der am häufigsten verwendeten Techniken in JavaScript. Dieses Skript verwendet wie das AppleScript oben zwei Variablen, `myLayerCount` und `myCounter`, um eine Ebene zu kennzeichnen und die Ebenen zu nummerieren, bis alle Ebenen im Dokument gekennzeichnet sind. Die Erhöhung findet in der dritten Anweisung innerhalb der Anweisung `for` statt: `myCounter++`. Die Syntax `++` fügt dem aktuellen Wert 1 hinzu; sie fügt jedoch erst 1 hinzu, wenn die Schleifenaktion ausgeführt wurde.

Die Schleife `for` in diesem Skript drückt Folgendes aus:

1. Mit dem Wert von `myCounter` bei 0 beginnen.
2. Wenn der Wert von `myCounter` kleiner ist als der Wert von `myLayerCount`, soll der Wert von `myCounter` als Index für die Ebene verwendet werden, die zu `myLayer` zugewiesen wurde und die Sichtbarkeit von `myLayer` soll auf `true` festgelegt werden.
3. Dem Wert von `myCounter` soll 1 hinzugefügt werden und der neue Wert von `myCounter` soll mit dem Wert von `myLayerCount` verglichen werden.
4. Wenn `myCounter` immer noch kleiner ist als `myLayerCount`, soll der neue Wert von `myCounter` als Index von `myLayer` verwendet werden und die Sichtbarkeit von `myLayer` auf `true` festgelegt und dann 1 zum Wert von `myCounter` hinzugefügt werden.
5. Der Vorgang soll so lange wiederholt werden, bis `myCounter` nicht mehr kleiner als `myLayerCount` ist.

## VBS

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.ActiveDocument
For Each object in myDoc.Layers
    object.Visible = True
Next
```

Die Schleife `For Each Next` in VBScript teilt der Anwendung mit, dass die Eigenschaft `Visible` jedes Objekts in der Kollektion `Layers` im aktiven Dokument auf `True` gesetzt werden soll. Beachten Sie, dass die Kollektion durch die Container-Hierarchie der übergeordneten Objekte gekennzeichnet wird (in diesem Fall durch die Variable `myDoc`), gefolgt von dem Kollektionsnamen, der die Pluralform des Objektnamens ist (in diesem Fall `Layers`).

**HINWEIS:** Bei dem in der Schleife benannten Objekt kann es sich um ein beliebiges Objekt handeln. Das Skript funktioniert ebenso, wenn Sie `object` durch `x` wie im folgenden Skript ersetzen:

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.ActiveDocument

For Each x in myDoc.Layers
    x.Visible = True
Next
```

## Weitere Informationen über Skripte

Jede Skriptsprache umfasst erheblich mehr Mittel und Techniken, mit denen Sie wichtige und komplexe Funktionen durch Skripte ausführen können. Weitere Einzelheiten, wie Sie mit Skripten Ihre Adobe-Anwendung steuern können, finden Sie im Adobe-Skripthandbuch für Ihre Anwendung. Referenzen finden Sie außerdem in [Kapitel 6, „Bibliographie“](#).

# 5 Fehlerbehebung

In diesem Kapitel wird beschrieben, wie Sie allgemeine Fehlermeldungen interpretieren, die beim Ausführen eines Skripts angezeigt werden können.

## Reservierte Wörter

Der Skripteditor und das ESTK sowie viele andere Skripteditoren markieren bestimmte Wörter, die Sie eingeben.

Beispielsweise werden die Booleschen Werte `true` und `false` stets markiert. Andere Beispiele werden nachfolgend aufgeführt.

**AS**

```
tell
end
with
set
```

**JS**

```
var
if
else
with
```

**VBS**

```
Dim
Set
MsgBox
```

Diese markierten Wörter sind von der Skriptsprache für besondere Zwecke reserviert und können nicht als Variablennamen verwendet werden. Sie können reservierte Wörter als Teil einer Zeichenfolge verwenden, da sie in Anführungszeichen eingeschlossen sind. Sie können sie auch in Kommentaren verwenden, weil Kommentare durch die Skript-Engine ignoriert werden.

Prüfen Sie bei einem Hinweis auf einen Syntaxfehler, ob Sie versehentlich ein reserviertes Wort verwendet haben. Eine vollständige Liste der in der Skriptsprache reservierten Wörter finden Sie in einer der Ressourcen in [Kapitel 6, „Bibliographie“](#).

## Fehlermeldungen des AppleScript-Skripteditors

Wenn das AppleScript-Skript einen Fehler enthält, markiert der Skripteditor den entsprechenden Teil des Skripts und zeigt eine Fehlermeldung an.

Prüfen Sie den markierten Teil des Skripts auf korrekte Rechtschreibung und Zeichensetzung. Wenn Sie keinen Fehler im markierten Text finden, prüfen Sie den Text direkt vor der Markierung. Wenn dieser Text einen Fehler enthält, hat die Skript-Engine möglicherweise einen anderen als den markierten Text erwartet.

Einige häufig vorkommende Fehlermeldungen sind nachfolgend erklärt.

- ▶ **Can't get object** – In der Regel haben Sie das Objekt in der Container-Hierarchie nicht richtig definiert. Versuchen Sie, das *parent-object* (wobei das *parent-object* das Objekt ist, das das in der Fehlermeldung aufgeführte Objekt enthält) nach dem Objektnamen im Skript hinzuzufügen oder erstellen Sie eine verschachtelte *tell*-Anweisung, die das übergeordnete Objekt benennt.
- ▶ **Expected "" but found end of script** – Vergewissern Sie sich, dass alle Anführungszeichen um Zeichenfolgen geschlossen sind.
- ▶ **Requested property not available for this object** – Prüfen Sie die Schreibweise aller Eigenschaften.

**TIPP:** Wählen Sie **Result Log** am Ende des Skripteditor-Fensters, um den Fortschritt des Skripts zeilenweise anzuzeigen.

## ESTK-Fehlermeldungen

Das ESTK macht Sie auf Fehler auf mehrere Arten aufmerksam:

- ▶ Wenn das Skript Syntaxfehler enthält, wird es nicht ausgeführt und der fehlerhafte Abschnitt des Skripts wird grau markiert. Häufig wird eine Beschreibung des Problems in der Statusleiste unten im ESTK-Fenster angezeigt.

Wenn ein Syntaxfehler vorliegt, prüfen Sie die folgenden Punkte:

- ▷ Vergewissern Sie sich, dass die Groß-/Kleinschreibung richtig ist. Alle Ausdrücke in JavaScript (mit Ausnahme von Aufzählungsnamen) beginnen mit Kleinbuchstaben und verwenden Großbuchstaben für den ersten Buchstaben in jedem Wort eines zusammengesetzten Ausdrucks, wie `artLayer`.

Außerdem ist bei Variablennamen auf die Groß-/Kleinschreibung zu achten.

- ▷ Schließen Sie alle runden und geschweiften Klammern sowie Anführungszeichen. Achten Sie darauf, dass Sie diese Zeichen paarweise verwenden.
- ▷ Achten Sie darauf, dass Sie gerade Anführungszeichen verwenden. Mischen Sie außerdem nicht einzelne und doppelte Anführungszeichen. Beispiel:

Falsch: `myDoc.name = "Eigenes Dokument"`

Richtig: `myDoc.name = 'Eigenes Dokument'`

Richtig: `myDoc.name = "Eigenes Dokument"`

**HINWEIS:** Manche Syntaxfehler wie typografische Anführungszeichen werden rot markiert. In der Statuszeile wird die Meldung „Syntax error“ angezeigt. Vergewissern Sie sich, dass Sie gerade Anführungszeichen verwenden.

- ▶ Wenn das Skript einen Laufzeitfehler enthält, wie ein Objekt, das nicht richtig gekennzeichnet ist oder eine Eigenschaft, die für das Objekt, das diese Eigenschaft verwenden möchte, nicht vorhanden ist, wird die fehlerhafte Anweisung hervorgehoben. Das Skript wird jedoch ausgeführt, wie durch das wirbelnde Symbol in der rechten unteren Ecke angezeigt wird. Der Fehler wird außerdem sowohl im Bereich der JavaScript Console als auch in der Statuszeile beschrieben.

Wenn ein Laufzeitfehler auftritt:

- ▷ Wählen Sie **Debug > Stop** oder drücken Sie die **Umschalttaste + F5**, um das Skript zu beenden.
- ▷ Sehen Sie in der JavaScript Console nach, um welche Art von Fehler es sich handelt. Die folgende kurze Beschreibung einiger häufig vorkommenden Fehlermeldungen kann Ihnen einen Anhaltspunkt für die Fehlerbehebung liefern.

*element is undefined* – Vergewissern Sie sich, wenn es sich bei dem nicht definierten Element um eine Variable handelt, dass der Variablenname richtig geschrieben ist und die richtige Groß-/Kleinschreibung verwendet wird. Vergewissern Sie sich außerdem, dass die Variable entweder mit einer Anweisung `var` definiert oder ihr ein Wert zugewiesen wurde.

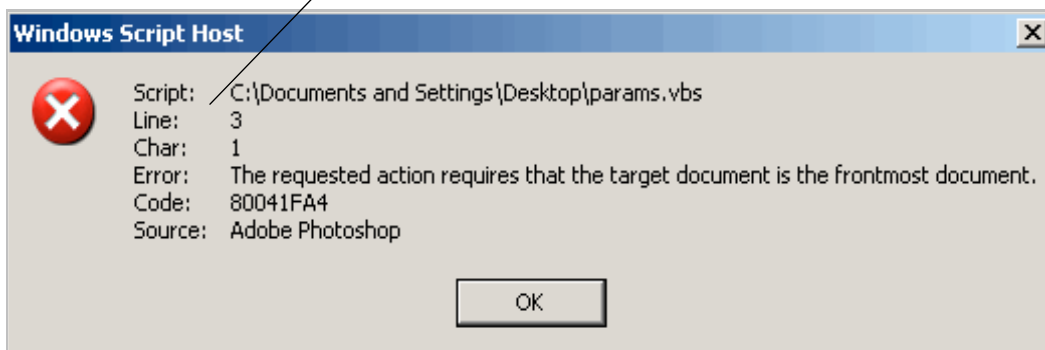
Wenn es sich bei dem nicht definierten Element um einen Zeichenfolgenwert handelt, vergewissern Sie sich, dass der Wert in Anführungszeichen gesetzt ist.

*undefined is not an object* – Vergewissern Sie sich, dass das Objekt in der markierten Anweisung richtig in der Container-Hierarchie gekennzeichnet wurde. Beispiel: Vergewissern Sie sich, wenn es sich bei dem Objekt um eine Ebene handelt, dass Sie das Dokument definiert haben, das die Ebene enthält. Bei Dokumentobjekten kann es erforderlich sein, die `app` des übergeordneten Objekts anzugeben.

## VBScript-Fehlermeldungen

Wenn das VBScript einen Fehler enthält, zeigt ein Windows Script Host eine Fehlermeldung an, die die Zeile kennzeichnet, in der der Fehler aufgetreten ist sowie die Position innerhalb der Zeile, an der die fehlerhafte Syntax bzw. das Objekt beginnt.

Diese Meldung weist darauf hin, dass sich das Problem am Beginn der 3. Zeile im Skript befindet



# 6 Bibliographie

Dieses Kapitel enthält eine Liste von Skriptbüchern für Anfänger. Diese Liste ist nicht vollständig. Online-Tutorials für Ihre Skriptsprache finden Sie außerdem im Internet.

## AppleScript

Weitere Informationen und Anweisungen über die AppleScript-Skriptsprache finden Sie in den folgenden Dokumenten und Ressourcen:

- ▶ *AppleScript for the Internet: Visual QuickStart Guide*, 1. Ausgabe, Ethan Wilde, Peachpit Press, 1998. ISBN 0-201-35359-8.
- ▶ *AppleScript Language Guide: English Dialect*, 1. Ausgabe, Apple Computer, Inc., Addison-Wesley Publishing Co., 1993. ISBN 0-201-40735-3.
- ▶ *Danny Goodman's AppleScript Handbook*, 2. Ausgabe, Danny Goodman, iUniverse, 1998. ISBN 0-966-55141-9.
- ▶ Apple Computer, Inc. AppleScript-Website:  
<http://www.apple.com/de/macosex/features/applescript/>

## JavaScript

Weitere Informationen und Anweisungen über die JavaScript-Skriptsprache finden Sie in den folgenden Dokumenten und Ressourcen:

- ▶ *JavaScript: The Definitive Guide*, David Flanagan, O'Reilly Media Inc, 2002. ISBN 0-596-00048-0.
- ▶ *JavaScript Bible*, Danny Goodman, Hungry Minds Inc, 2001. ISBN 0-7645-4718-6.
- ▶ *Adobe Scripting*, Chandler McWilliams, Wiley Publishing, Inc., 2003. ISBN 0-7645-2455-0.

## VBScript

Weitere Informationen und Anweisungen über die VBScript- und VBSA-Skriptsprache finden Sie in den folgenden Dokumenten und Ressourcen:

- ▶ *Learn to Program with VBScript 6*, 1. Ausgabe, John Smiley, Active Path, 1998. ISBN 1-902-74500-0.
- ▶ *Microsoft VBScript 6.0 Professional*, 1. Ausgabe, Michael Halvorson, Microsoft Press, 1998. ISBN 1-572-31809-0.
- ▶ *VBS & VBSA in a Nutshell*, 1. Ausgabe, Paul Lomax, O'Reilly, 1998. ISBN 1-56592-358-8.
- ▶ Skript-Website von Microsoft Developers Network (MSDN):  
[msdn.microsoft.com/scripting](http://msdn.microsoft.com/scripting)

# Index

## A

- Aktionen, 6
- AppleScript
  - Definition, 6
  - erstes Skript, 8
  - Funktionsverzeichnisse, 33
  - Website, 53
- Argumente
  - Definition, 9
  - verwenden, 25
- Arrays, 31
  - definierte, 17
  - erstellen, 31
- Aufzählungen
  - definierte, 17
  - verwenden, 22

## B

- Befehle
  - anzeigen in AS-Funktionsverzeichnissen, 33, 35
  - Eigenschaften, 24, 25
  - verwenden, 24
- Bibliographie, 53
- Boolesch, 17

## C

- Container-Hierarchie, 10, 12
  - in Skriptreferenzen, 42

## D

- Datentypen, 16
- Dialogfelder, 20
- DOM
  - anzeigen, 10
  - Definition, 10

## E

- Eigenschaften
  - anzeigen in AS-Funktionsverzeichnissen, 33
  - anzeigen in Skriptreferenzen, 42
  - anzeigen in VBS-Typbibliotheken, 38
  - Datentypen, 16
  - Definition, 9

- Lesen/Schreiben, 20
  - mehrere Werte, 17
  - schreibgeschützt, 20
  - verwenden, 16

## Elemente

- anzeigen in Skriptreferenzen, 42

## ESTK

- Fehlerbehebung in, 51
- JS-Objektmodell anzeigen, 36
- Standardspeicherort, 7

## ExtendScript

- Definition, 7

## F

- Funktionsverzeichnisse, 33

## I

- if else -Anweisungen, 47
- if-Anweisungen, 46
- Illustrator, *Siehe* Adobe Illustrator
- Index
  - Definition, 13
  - Nummerierungsschemata, 14

## J

- JavaScript
  - Definition, 7
  - erstes Skript, 8
  - Groß-/Kleinschreibung, 15
  - Vorteile, 7
- JavaScript Tools Guide, 7

## K

- Kommentare, 29
- Konditionalanweisungen, 46
- Konstante
  - definierte, 17
  - verwenden, 22

## L

- Lange Zeilen, 30

**M**

Makros, 6

Methoden

- anzeigen in Skriptreferenzen, 44
- anzeigen in VBS-Typbibliotheken, 40
- Argumente, 25
- Definition, 9
- verwenden, 24

**O**

Objekte

- aktive, 15
- aktuelle, 15
- anzeigen in AS-Funktionsverzeichnissen, 33, 35
- anzeigen in Skriptreferenzen, 42
- anzeigen in VBS-Typbibliotheken, 37
- Definition, 9
- Elemente, 13
- Kollektionen, 13
- übergeordnete, 10
- Verweise, 10
- verwenden, 9

**P**

Parameter

- anzeigen in Skriptreferenzen, 44
- bezeichnete (AS), 26
- Definition, 9
- direkt (AS), 26
- erforderlich, 25
- mehrere verwenden, 26
- optional, 25
- verwenden, 25

**S**

Schleifen, 48

Skripte

- automatisch ausführen, 7

Skripteditor

- AppleScript-Funktionsverzeichnisse, 33
- Fehlerbehebung in, 50
- Standardspeicherort, 6

Skriptkommentare, 29

Skripts

- Definition, 6
- Einführung, 6
- verwenden, 5

Startskript-Ordner, 7

**T**

tell-Anweisungen (AS), 27

**U**

Übergeordnetes Objekt, 10

**V**

var, 11

Variablen

- als Eigenschaftswerte, 17
- benennen, 13
- Definition, 11
- erstellen, 11
- für vorhandene Objekte, 29
- verwenden für Eigenschaftswerte, 24
- Wert ändern, 28
- Wertdefinition, 11

VBScript

- Definition, 7
- erstes Skript, 8
- Erweiterung, 8
- Typbibliotheken, 36

**W**

Warnfelder, 20

**Z**

Zeichenfolgen, 16