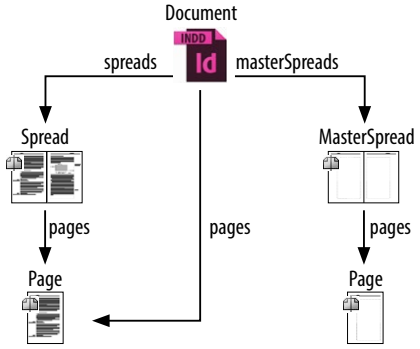
7.7 Seiten und Mustervorlagen

Wenn man mit InDesign arbeitet, ist die Unterscheidung zwischen Druckbogen, Mustervorlagen und Seiten klar, beim Skripting sollte man die Hierarchie kennen.

Abb. 62
Dokument, Seiten und Musterseiten



Die Abbildung zeigt, dass es eigentlich die klare Hierarchie Dokument, Druckbogen, Seite gibt. Das Objektmodell erlaubt jedoch den direkten Zugriff auf die Seiten über das Objekt `Document`. Diese Abkürzung wird oft verwendet, weil die Adressierung über das Objekt `Spread` unübersichtlicher ist und zu aufwändigerem Code führt. Außerdem beinhaltet das Objekt `Spread` auch die Montagefläche und Objekte, die sich darauf befinden.

Für Musterseiten gibt es kein Objekt `MasterPage`, die Druckbögen der Mustervorlagen `MasterSpread` enthalten normale Seiten. Musterseiten werden durch ihre Zugehörigkeit zu einem Musterdruckbogen gekennzeichnet. Deswegen muss man Musterseiten immer über die Eigenschaft `masterSpreads` adressieren. Dies führt leider manchmal zu Verwirrung.

Umgang mit Seiten

Im Folgenden werden die wichtigsten Eigenschaften der Klasse `Page` vorgestellt. Mit der Eigenschaft `name` kann die Seitenzahl der Seite ermittelt werden. Die Eigenschaft enthält einen String, der für die Adressierung in der Sammlung `Pages` mit der Methode `itemByName()` verwendet werden kann. Achten Sie darauf, dass es sich hier bei einer manuellen Seitennummerierung nicht unbedingt um die Position der Seite im Dokument handelt.

Position der Seite mit Dokument

Die Position im Dokument – und damit auch den Index für die Adressierung mit den eckigen Klammern – ermittelt man mit der Eigenschaft `documentOffset`.

Mit `side` kann man herausfinden, ob es sich um die rechte (`PageSideOptions.RIGHT_HAND`) oder linke Seite (`PageSideOptions.LEFT_HAND`) einer Doppelseite oder um eine Einzelseite (`PageSideOptions.SINGLE_SIDED`) handelt.

```
1  var _page = app.activeDocument.pages[0];
2  var _seite;
3  if (_page.side == PageSideOptions.LEFT_HAND) _seite = "linke";
4  else if (_page.side == PageSideOptions.RIGHT_HAND) _seite =
   "rechte";
5  else _seite = "einzelne";
6  alert("Position:" + _page.documentOffset + " , Name (Seitenzahl): "
   + _page.name + ",Typ: " + _seite + " Seite");
```

Listing 65
Eigenschaften der Seite
7-7_Seiten.jsx

Beachten Sie, dass ich in den Zeilen 3–5 die geschweiften Klammern der if- und else-Blöcke weggelassen habe. Die geschweiften Klammern würden hier den Code eher unübersichtlich machen. Wenn Sie wollen, können Sie sie natürlich hinzufügen.

Umgang mit Mustervorlagen

Auf Mustervorlagen sollten die feststehenden und/oder wiederkehrenden Objekte des Layouts platziert werden. Ähnlich wie Formate erstelle ich die benötigten Mustervorlagen nicht per Skript, sondern baue diese in InDesign. Das Dokument mit meinen Mustervorlagen verwende ich dann im Skript als Vorlage. Wenn man in den MUSTERVORLAGENOPTIONEN eindeutige Namen vergeben hat, können diese im Skript mit der Methode `itemByName()` adressiert werden.

Beachten Sie, dass sich der Name aus dem Namen und dem Präfix, so wie er auch im Bedienfeld SEITEN erscheint, zusammensetzt.

Um eine Mustervorlage auf eine Seite anzuwenden, muss die Mustervorlage der Eigenschaft `appliedMaster` zugewiesen werden.

Das folgende Beispiel setzt eine Mustervorlage mit der Präfix-Namen-Kombination `A-Mustervorlage` voraus. Das Beispieldokument *7-7_Mustervorlagen.idml* ist für die Verwendung mit den restlichen Skripten des Unterkapitels vorbereitet.

```
1  var _dok = app.activeDocument;
2  var _musterVorlage = _dok.masterSpreads.
   itemByName("A-Mustervorlage");
3  _dok.pages[0].appliedMaster = _musterVorlage;
```

Listing 66
Adressierung von Mustervorlagen
7-7_Mustervorlagen-1.jsx

Mustervorlagenobjekte mit `override()` lösen

Mustervorlagenobjekte sind auf der Seite im Layout normalerweise gesperrt, dies erkennt man an den gepunkteten Rahmenkanten. Die Idee ist, dass das Objekt nicht zur eigentlichen Seite, sondern zur Musterseite gehört. Wenn Sie z. B. eine Grafik auf der Musterseite platzieren, wird diese bei 50 Seiten nicht 50-mal platziert, sondern nur einmalig auf der Musterseite. Trotzdem kommt es vor, dass Objekte auf der Musterseite gelöst werden müssen. In InDesign klickt man dazu mit gedrückter SHIFT + BEFEHLSTASTE auf das entsprechende Objekt. Im Skripting kann dazu die Methode `override()` verwendet werden. Die Methode erwartet als Parameter die Seite, auf der das Objekt gelöst werden soll.

Im folgenden Beispiel wird das zuletzt erstellte Mustervorlagenobjekt der auf der ersten Seite des Dokuments angewendeten Mustervorlage gelöst.

Listing 67
Lösen eines Mustervorlagenobjekts
7-7_Mustervorlagen-2.jsx

```
1  var _dok = app.activeDocument;
2  var _page = app.activeDocument.pages[0];
3  var _musterVorlage = _dok.masterSpreads.
   itemByName("A-Mustervorlage");
4  _page.appliedMaster = _musterVorlage;
5  _page.masterPageItems[0].override(_page);
```

Nachdem in der Variablen `_page` die erste Seite des Dokuments gespeichert wurde, wird ihr in Zeile 3 die Mustervorlage `A-Mustervorlage` zugewiesen.

5 Vom Objekt `Page` kann man mit der Eigenschaft `masterPageItems` auf die Mustervorlagenobjekte zugreifen, die auf der Seite enthalten sind. Als Rückgabewert bekommt man einen Array; um das Objekt zu lösen, muss man den Index des Objekts kennen. Mit der Methode `override()` wird das Objekt auf der Seite, die als Parameter übergeben wird, gelöst – in diesem Fall also auf der ersten Seite, die über `_page` referenziert wurde.

Da in der Eigenschaft `masterPageItems` ein Array gespeichert ist, kann man nicht mit den Methoden der Sammlungen wie `itemByName()` auf die Mustervorlagenobjekte zugreifen. Da sich der Index innerhalb des Arrays `masterPageItems` bei der Erstellung von neuen Objekten ändert, ist der Zugriff über die Eigenschaft `masterPageItems` nur in Ausnahmefällen empfehlenswert. Besser ist es, die Objekte auf der Musterseite mit Namen zu versehen und diese dann zu lösen.

Listing 68
Lösen von Mustervorlagenobjekten
7-7_Mustervorlagen-3.jsx

```
5  var _vorlagenObjekt = _musterVorlage.pageItems.itemByName("tf");
6  _vorlagenObjekt.override(_page);
```

5 Anstatt über die Seite wird das Objekt auf der eigentlichen Musterseite adressiert. Dort muss natürlich ein Objekt mit dem Namen `tf` vorhanden sein. Wie man Objekte mit Namen versieht, wurde auf Seite 86 besprochen.

6 Das in `_vorlagenObjekt` gespeicherte Musterseitenobjekt kann jetzt mit der Methode `override()` gelöst werden. Hier wird auch deutlicher, dass als Parameter die Seite, auf der das Objekt gelöst werden soll, übergeben werden muss.

++ Musterseitenobjekte sperren

Rahmen auf Musterseiten können gesperrt werden, so dass sie nicht mehr auf der Seite, auf der die Musterseite angewendet wurde, gelöst werden können. Dazu muss man per Skript die Eigenschaft `allowOverrides` auf den Wert `false` setzen.

Primäre Textrahmen und automatischer Textumfluss

Wenn ein Textrahmen immer gelöst werden soll, beispielsweise weil ein Text platziert und automatisch umbrochen werden soll, muss auf

der Musterseite ein primärer Textrahmen eingerichtet werden. Dieser wird automatisch gelöst, wenn die Musterseite verwendet wird.

Einrichtung eines primären Textrahmens

Einen primären Textrahmen in Satzspiegelgröße können Sie bei der Erstellung des Dokuments von InDesign hinzufügen lassen. Wählen Sie dazu die Option PRIMÄRER TEXTRAHMEN im Dialog NEUES DOKUMENT an. Alternativ können Sie nachträglich einen Textrahmen umwandeln, indem Sie mit der Maus auf das Symbol in der linken oberen Ecke klicken. Das Symbol wechselt dann zu , so dass der primäre Textrahmen leicht erkennbar ist. Beachten Sie, dass der Textrahmen leer sein muss. Per Skript könnten Sie in der Eigenschaft `primaryTextFrame` der Musterseite einen Textrahmen speichern, in der Praxis ist es aber leichter, dies in der Arbeitsvorbereitung manuell zu erledigen!

Automatischer Umbruch

Für einen automatischen Umbruch muss die Voreinstellung INTELLIGENTER TEXTUMFLUSS aktiviert werden. Diese Einstellung finden Sie unter BEARBEITEN → VOREINSTELLUNGEN→ EINGABE.... Wenn der primäre Textrahmen bei der Bearbeitung einen Textüberlauf bekommt, werden automatisch Seiten hinzugefügt. Zusammen mit der Funktion `place()` aus dem vorherigen Unterkapitel können Sie einen automatischen Seitenumbruch skripten.

Listing 69
Intelligenten Textumfluss per Skript aktivieren
7-7_Textumfluss.jsx

```
1  var _dok = app.activeDocument;
2  with (_dok.textPreferences){
3    smartTextReflow = true;
4    limitToMasterTextFrames = true;
5    addPages = AddPageOptions.END_OF_STORY;
6    preserveFacingPageSpreads = true;
7    deleteEmptyPages = false;
8  }
```

2–8 Mit Hilfe eines with-Statements werden die Einstellungen für den intelligenten Textumfluss aktiviert. Das Ergebnis entspricht den Einstellungen in der folgenden Abbildung:

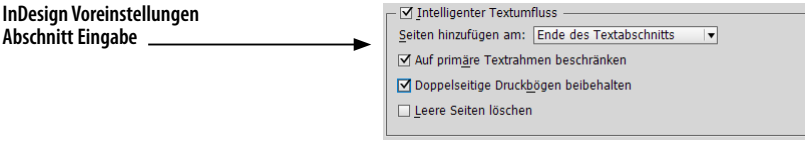


Abb. 63
Intelligenter Textumfluss

7.8 Rahmen und Seitenobjekte

Wenn man die Seiten gemeistert hat, trifft man auf die Objekte, die auf ihnen platziert sind. Dazu zählen Rechtecke, Textrahmen, Ellipsen, Linien sowie Gruppen und einige mehr – letztlich alle Objekte, die auf einer Seite platziert sein können. Alle Seitenobjekte sind von der Basisklasse `PageItem` abgeleitet.

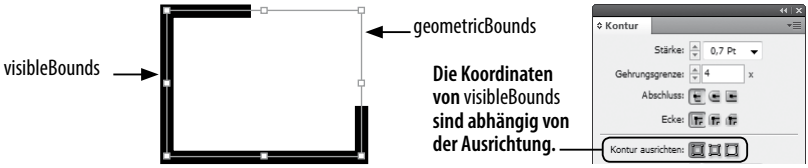
Tab. 17
Wichtige, von `PageItem` abgeleitete Objekte

InDesign-Objekt	Skripting-Objekt
Textrahmen	`TextFrame`
Rechteckrahmen	`Rectangle`
Ellipsenrahmen	`Oval`
Grafik	`Graphic`
Linie	`GraphicLine`
Gruppe	`Group`
Schaltfläche	`Button`

Der Umgang mit all diesen Objekten ist recht ähnlich; die wichtigsten Eigenschaften werden gleich vorgestellt.

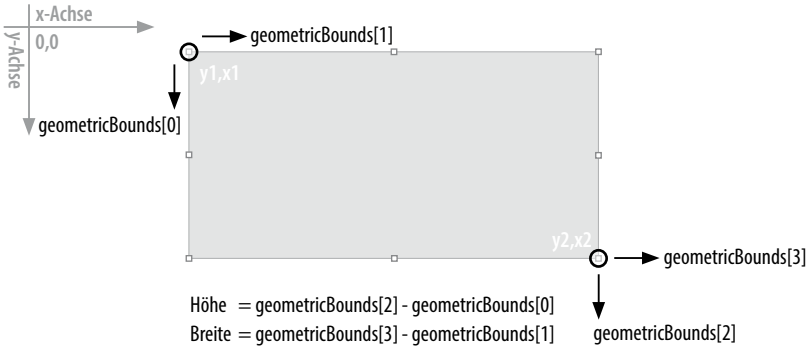
Mit der Eigenschaft `geometricBounds` bzw. `visibleBounds` wurden bereits Größen und Positionen von Seitenobjekten ausgewertet. Während Erstere die Rahmenkanten repräsentieren, enthalten Letztere die Größe des Rahmens inklusive der Rahmenkontur. Die folgende Abbildung zeigt den Zusammenhang zwischen der Konturposition und den Werten der beiden Eigenschaften.

Abb. 64
Unterschied zwischen geometricBounds und visibleBounds



Beide Eigenschaften enthalten einen Array mit vier Koordinaten. Das Problem beim Skripting ist meist, dass man die Ansicht und die Werte aus dem Bedienfeld Steuerung gewohnt ist. Hier werden die x- und y-Position des Rahmens sowie Breite und Höhe angezeigt. In der Eigenschaft `geometricBounds` bzw. `visibleBounds` finden sich nur vier Koordinaten, die außerdem noch vertauscht sind. Die ersten beiden Werte definieren die linke obere Ecke des Rahmens, die letzten beiden Werte die rechte untere Ecke des Rahmens. Es wird immer zuerst die vertikale Koordinate (y-Wert) angegeben, danach folgt die horizontale Koordinate (x-Wert).

Abb. 65
Koordinaten von Seitenobjekten



Höhe und Breite kann man sich leicht errechnen:

```
1  var _ausw = app.selection[0];
2  var _hoehe = _ausw.geometricBounds[2] - _ausw.geometricBounds[0];
3  var _breite = _ausw.geometricBounds[3] - _ausw.geometricBounds[1];
4  alert("Das Objekt hat die Höhe " + _hoehe + " und Breite " + _breite);
```

Listing 70
Höhe und Breite eines Seitenobjekts berechnen
7-8_HoeheBreite.jsx

Ein Objektformat verwenden

Rahmen und Seitenobjekte haben viele Attribute bezüglich des Aussehens, die Details finden Sie im Objektmodell. Oft verwendet werden `strokeWeight` für die Konturenstärke, `strokeColor` für die Farbe der Kontur, `fillColor` für die Hintergrundfarbe. Alle diese Eigenschaften können auch in Objektformaten hinterlegt werden.

Bei der Einrichtung von Objektformaten gibt es die Möglichkeit, Bereiche mit bestimmten Attributen wie Fläche, Kontur oder Transparenz zu deaktivieren. Bei der Anwendung des Formats auf ein Seitenobjekt werden die deaktivierten Attribute unverändert beibehalten, entsprechend wird auch keine Abweichung zum Format im Bedienfeld angezeigt. Die nicht vom Format definierten Attribute können mit der Schaltfläche am unteren Rand des Bedienfelds Objektformate gelöscht werden. Die Attribute werden dann auf die Werte des Objektformats [Ohne] gesetzt.

```
var _of = app.activeDocument.objectStyles.itemByName("NameOF");
app.selection[0].applyObjectStyle(_of, false, false);
```

Objektformat zuweisen

Ein Objektformat weisen Sie am besten mit der Methode `applyObjectStyle()` zu. Ihr wird als erster Parameter das Objektformat übergeben. Mit dem zweiten Parameter kann gesteuert werden, ob Formatabweichungen des Objekts gelöscht werden. Mit dem Wert `true` werden alle abweichenden Formatattribute entfernt, bei `false` bleiben diese erhalten. Mit dem dritten Parameter können Sie die nicht vom Format definierten Attribute löschen bzw. erhalten. Mit `true` werden sie gelöscht, bei `false` nicht. Der zweite und dritte Parameter sind optional, der Standardwert für den zweiten ist `true`, für den dritten `false`.

```
app.selection[0].appliedObjectStyle = app.activeDocument.objectStyles.
   itemByName("NameOF");
```

Alternativ könne Sie auch mit der Eigenschaft `appliedObjectStyle` das Objektformat festlegen, dann werden die Abweichungen zum Format gelöscht, aber die nicht vom Format definierten Attribute beibehalten.

Weitere wichtige Eigenschaften

Skalierung und Drehung

Die Drehung und Skalierung von Seitenobjekten sollen oft unabhängig von einem Objektformat eingerichtet werden. Für die Skalierung kommen die Eigenschaften `absoluteHorizontalScale` und `absoluteVerticalScale` zum Einsatz. Beiden wird der gewünschte Pro-

zentwert als Zahl übergeben. Bei der Skalierung stehen zwei Eigenschaften zur Verfügung, denen beliebige Winkel von –360° bis 360° als Zahl übergeben werden können. Mit `absoluteRotationAngle` kann der Drehwinkel relativ zum Elternobjekt angegeben werden. Mit der Eigenschaft `rotationAngle` wird der eigentliche Drehwinkel des Objekts angegeben – der Wert, den Sie auch in der Benutzeroberfläche sehen. Zur Verdeutlichung kann man sich eine in einem Rechteckrahmen platzierte Grafik vorstellen: Wenn diese um 10° gedreht wird, enthält die Eigenschaft `absoluteRotationAngle` des Rechtecks die Zahl 10, die der Grafik 0. Der Wert der Eigenschaft `rotationAngle` ist in beiden Fällen 10.

Zugriff auf alle Seitenobjekte

Alle Seitenobjekte eines Dokuments, einer Seite oder einer Ebene erhalten Sie über die Eigenschaft `allPageItems`, die einen Array mit den Seitenobjekten enthält. Da in Seitenobjekten weitere Seitenobjekte enthalten sein können, z. B. in einer Gruppe, hat auch die Klasse `PageItem` die Eigenschaft `allPageItems`. Zusätzlich haben diese Objekte auch die Eigenschaft `pageItems`, in der sich die Sammlung der Seitenobjekte befindet. Die Sammlung unterscheidet sich vom Array darin, dass in der Sammlung die Hierarchieebenen von gruppierten, verankerten oder ineinanderkopierten Seitenobjekten abgebildet sind. Der Inhalt der Sammlung entspricht einer Hierarchieebene im Bedienfeld Ebenen. Der Vorteil der Sammlung ist, dass mit der Methode `itemByName()` ein benanntes Objekt adressiert werden kann.

Eltern von Seitenobjekten

Wenn Sie den Rahmen nicht durch die klassische Hierarchie über die Seite adressiert haben, kommen Sie mit der Eigenschaft `parentPage` auf die Seite, auf der sich der Rahmen befindet. Wenn sich der Rahmen über zwei Seiten erstreckt, wird die Seite mit der größeren Fläche zurückgeliefert. Wenn sich der Rahmen auf der Montagefläche befindet, liefert die Eigenschaft `null`. Die Eigenschaft `parent` führt vom Rahmen zum Druckbogen. Ein klassischer Anwendungsfall ist die automatische Platzierung von Bildern, deren Dateinamen im Manuskript enthalten sind. Hier sucht man die entsprechenden Textstellen und ermittelt dann über `parentPage` die Seite, auf der das Bild platziert werden soll (→ Unterkapitel 12.4). Für Preflight- oder Analyse-Skripte ist es ebenfalls hilfreich, die Seite, auf der ein Fehler enthalten ist, zu kennen.

Konturenführung

Wenn man die Konturenführung nicht über ein Objektformat festlegen will, kann man dies auch direkt per Skript erledigen. Alle Seitenobjekte haben die Eigenschaft `textWrapPreferences`, in der die Einstellungen für die Konturenführung hinterlegt sind.

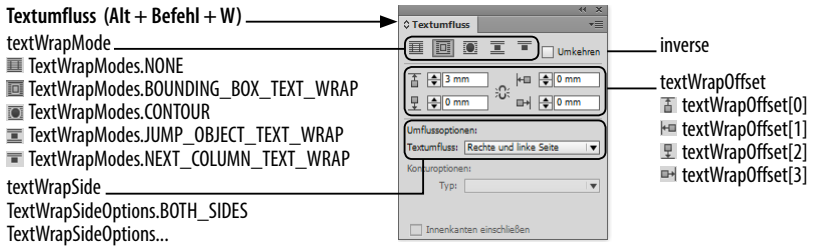


Abb. 66
Zusammenhang Konturenführung-Bedienfeld und Klasse textWrapPreferences

Das nächste Skript zeigt, wie man diese Einstellungen per Skript setzen kann. Dazu können Sie das Dokument *7-8_Konturenfuehrung.idml* verwenden. Vor der Ausführung muss der Bildrahmen ausgewählt werden.

Listing 71
7-8_Konturenfuehrung.jsx

```
var _auswahl = app.selection[0];
with (_auswahl.textWrapPreferences) {
  textWrapMode = TextWrapModes.BOUNDING_BOX_TEXT_WRAP;
  textWrapOffset = [3,3,0,3];
  textWrapSide = TextWrapSideOptions.BOTH_SIDES;
}
```

4 Die Angabe der Abstände mit `textWrapOffset` muss in Abhängigkeit des `textWrapMode` unterschiedlich vorgenommen werden. Im Fall `TextWrapModes.BOUNDING_BOX_TEXT_WRAP` wird ein Array mit vier Werten übergeben `[oben, links, unten, rechts]`. Für den Wert `TextWrapModes.JUMP_OBJECT_TEXT_WRAP` müssen zwei Werte im Format `[oben, unten]` angegeben werden. Im Fall `TextWrapModes.NEXT_COLUMN_TEXT_WRAP` und `TextWrapModes.CONTOUR` wird ein einzelner Wert ohne Array übergeben.

QR-Codes

Seit InDesign CC können in Seitenobjekten QR-Codes eingesetzt werden, üblicherweise werden sie in einem Rahmen platziert. Das folgende Skript erstellt einen QR-Code, der zur Webseite des Buches führt. Vor der Ausführung muss ein Rahmen ausgewählt werden.

Listing 72
7-8_QRCode.jsx

```
var _auswahl = app.selection[0];
var _url = "http://www.indesignjs.de";
var _swatch = app.activeDocument.swatches[3];
_auswahl.createHyperlinkQRCode( _url, _swatch);
```

4 Die Methode `createHyperlinkQRCode()` übernimmt zwei Parameter. Der erste enthält die URL zur Webseite. Mit dem zweiten, optionalen Parameter kann die Farbe des QR-Codes gesteuert werden. Beide Parameter wurden der Übersichtlichkeit halber zuvor in Variablen gespeichert, könnten aber auch direkt übergeben werden. Im Unterkapitel 14.3 finden Sie ein weiteres Beispiel im Praxiseinsatz.

Abb. 67
Der generierte QR-Code

Die folgenden QR-Code-Typen können erstellt werden. Die Parameter mit den Werten des QR-Codes werden als String übergeben. Die Angabe eines Farbfelds im letzten Parameter ist optional; wenn Sie keine Farbe übergeben, wird Schwarz verwendet.

Tab. 18 QR-Codes ab InDesign CC

Typ	Funktion und Parameter
E-Mail	`createEmailQRCode (emailAddress, subject, body, [qrCodeSwatch])`
Webadresse	`createHyperlinkQRCode (urlLink, [qrCodeSwatch])`
Text	`createPlainTextQRCode (plainText, [qrCodeSwatch])`
SMS	`createTextMsgQRCode(cellNumber, textMessage, [qrCodeSwatch])`
VCard	`createVCardQRCode (firstName, lastName, jobTitle, cell Phone, phone, email, organisation, streetAddress, city, adrState, country, postalCode, website, [qrCodeSwatch])`

Die Methoden `create...QRCode()` generieren jeweils ein Objekt vom Typ `EPS` in einem Rahmen. Die Eigenschaften des QR-Codes können nicht nachträglich angepasst werden. Wenn das nötig ist, kann man einfach einen neuen QR-Code erstellen.

7.9 Textrahmen

Textrahmen sind von der Klasse `PageItem` abgeleitet, weisen aber ein paar Besonderheiten auf. Aus InDesign kennt man die Möglichkeiten, Textrahmen zu verketten. Dazu besitzt die Klasse `TextFrame` die Eigenschaften `nextTextFrame` und `previousTextFrame`. Beiden kann man als Wert den Textrahmen, der verkettet werden soll, zuweisen. Für die folgenden Skripte kann das Dokument *7-9_Textrahmen.idml* verwendet werden.

Listing 73 Verkettung von zwei Textrahmen *7-9_Verketten.jsx*

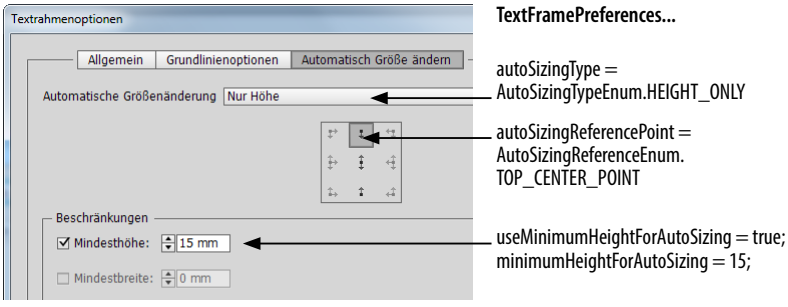
```
1  var _seite1 = app.activeDocument.pages[0];
2  var _seite2 = app.activeDocument.pages[1];
3  _seite1.textFrames[0].nextTextFrame = _seite2.textFrames[0];
```

3 Der Eigenschaft `nextTextFrame` des Textrahmens auf der ersten Seite wird der Textrahmen der zweiten Seite zugewiesen. Die beiden Rahmen sind nun verkettet.

Textrahmengröße automatisch festlegen

Ein interessante Möglichkeit von Textrahmen ist die Funktion Automatisch Grösse ändern, mit der die automatische Größenänderung gesteuert werden kann. Die Einstellungen können in den Textrahmenoptionen vorgenommen werden.

Abb. 68 Textrahmenoptionen für Automatisch Größe ändern



```
1  var textFrame = app.selection[0];
2  with (textFrame.textFramePreferences) {
3    autoSizingType = AutoSizingTypeEnum.HEIGHT_ONLY;
4    autoSizingReferencePoint =
     AutoSizingReferenceEnum.TOP_CENTER_POINT;
5    useMinimumHeightForAutoSizing = true;
6    minimumHeightForAutoSizing = 15;
7  }
```

Listing 74
Automatisch Größe ändern per Skript
7-9_Auto_Textrahmen.jsx

2–7 Wie üblich ist die grafische Benutzeroberfläche in einer Preferences-Klasse abgebildet, die einzelnen Eigenschaften befinden sich hier in `textFramePreferences`. Mit der Eigenschaft `autoSizingType` wird festgelegt, wie der Textrahmen angepasst werden soll. Entsprechend sind dann auch nicht immer alle Eigenschaften möglich. Im Falle der automatischen Höhenanpassung `AutoSizingTypeEnum.HEIGHT_ONLY` kann bspw. der `autoSizingReferencePoint` nur auf `AutoSizingReferenceEnum.TOP_CENTER_POINT`, `CENTER_POINT` bzw. `BOTTOM_CENTER_POINT` gesetzt werden.

Die Einstellungen können auch in einem Objektformat hinterlegt werden, was das oben gezeigte Skript überflüssig macht!

Komplexe Rahmen funktionieren nicht.

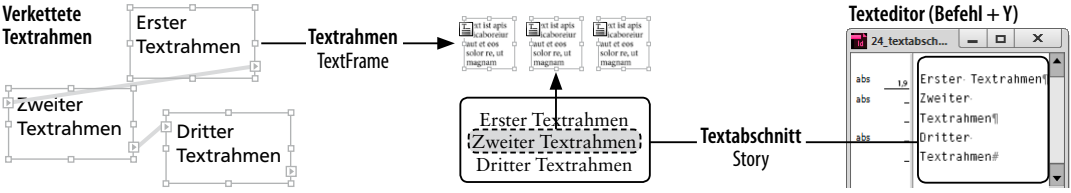
Wenn Sie die automatische Texteinpassung verwenden, sollten Sie auf die Funktion Spalten ausgleichen in Kombination mit der Absatzformatoption Spaltenspanne verzichten. Die Bearbeitung kann unangenehm langsam werden und InDesign CS6 sogar abstürzen.

Textrahmen vs. Text

Text und Textabschnitt

Der Textinhalt von einem bzw. der Textfluss von mehreren verketteten Textrahmen wird als *Textabschnitt* bezeichnet. Die zugehörige Klasse im Objektmodell heißt `Story`. Die Sammlung `Stories`, in der sich alle Textabschnitte des Dokuments befinden, gehört zum Dokument. Ein Textabschnitt enthält wie alle Textobjekte nicht nur den Inhaltstext als String, sondern alle Formatinformationen.

Abb. 69
Textabschnitte vs. Textrahmen



Ein Textabschnitt kann sich über mehrere Textrahmen (und Textpfade) erstrecken. Diese sind in einem Array gesammelt, den man über die Eigenschaft `textContainers` ansprechen kann. Verwechseln Sie `textContainers` nicht mit der ebenfalls vorhandenen Eigenschaft `textFrames`, hier sind die verankerten Textrahmen des Textabschnitts gespeichert.

Vom Textrahmen zum Textabschnitt

Wie aber kommt man nun vom Textrahmen zum Textabschnitt – die Eigenschaft `parent` führt ja zum Druckbogen? Für die Abbildung dieser Beziehung haben Textrahmen die Eigenschaft `parentStory`.

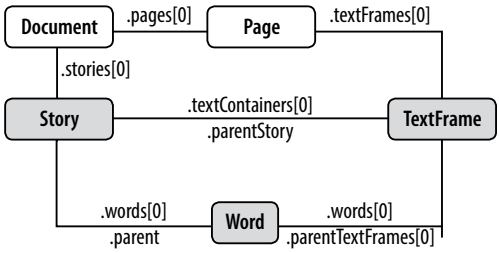
Vom Textobjekt zum Textrahmen

Die Textabschnitte untergliedern sich weiter in Absätze, Wörter und Zeichen. Diese Objekte haben als Elternobjekt den Textabschnitt, sind aber natürlich auch in Textrahmen platziert. Von Textobjekten, wie z. B. `paragraph`, kann man mit der Eigenschaft `parentTextFrames` zu dem oder den Textrahmen, die das Objekt enthalten, navigieren. Achten Sie darauf, dass hier immer ein Array enthalten ist, weil z. B. ein Absatz auf mehrere Textrahmen verteilt sein kann. Im Normalfall adressiert man den Textrahmen deswegen mit:

```
app.activeDocument.stories[0].words[2].parentTextFrames[0];
```

Da alle Textobjekte von der gleichen Klasse abgeleitet sind, gilt dies sogar für Zeichen und Einfügemarken, obwohl diese nur in einem Textrahmen stehen können.

Abb. 70 Klassenhierarchie von Textrahmen, -abschnitten und -objekten



Der folgende Code zeigt die Navigation zum Textrahmen, in dem das letzte Wort des Textabschnitts steht. Das Beispiel ist nicht praxisrelevant, zeigt aber die Zusammenhänge innerhalb der Hierarchie auf.

Listing 75 Navigation zum letzten Textrahmen der Story *7-9_NavigationText.jsx*

```
1  var _textRahmenSeite1 = app.activeDocument.textFrames[0];
2  var _textAbschnitt = _textRahmenSeite1.parentStory;
3  var _letztesWort = _textAbschnitt.words[-1];
4  var _letzterTextRahmen = _letztesWort.parentTextFrames[0];
```

Den letzten Textrahmen eines Textabschnitts adressieren

In der Praxis sollte man den letzten von mehreren verketteten Textrahmen über die Eigenschaft `endTextFrame` adressieren:

```
var _letzterTextRahmen = app.activeDocument.textFrames[0].endTextFrame;
```

Auf die Textobjekte Absatz, Wort, Zeichen kann man außerdem auch direkt vom Textrahmen zugreifen, was das Programmieren zwar einfacher, aber auch etwas unübersichtlicher macht.

Wenn man den Inhalt eines Textrahmens, der mit anderen Rahmen verknüpft ist, ermitteln will, kann die Eigenschaft `texts` verwendet werden. Die Klasse `Text` ist gleichzeitig die Basisklasse für alle Textobjekte.

Übersatz

Um wiederum herauszufinden, ob sich ein Text, der z. B. über die Suche lokalisiert wurde, im Übersatz befindet, prüft man die Eigenschaft `parentTextFrames`. Wenn der enthaltene Array keine Elemente hat, ist der Text nicht im Layout sichtbar.

```
var _letztesZeichen = app.activeDocument.stories[0].characters[-1];
if (_letztesZeichen.parentTextFrames.length == 0) { // Übersatz
```

Index

10 Noch mehr finden mit GREP

Neben den in Kapitel 1 gezeigten Techniken gibt es noch weitere Methoden, um mit GREP effektive Suchen und Ersetzungen vorzunehmen. Dieses Kapitel enthält einige spezielle Techniken, die ihre Stärken in bestimmten Situationen ausspielen. Die Anwendung zeige ich jeweils anhand eines konkreten Beispiels. Wer mit GREP-Stilen arbeitet, sollte besonderes Augenmerk auf Unterkapitel 10.6 legen. Im Unterkapitel 10.12 bis 10.15 sind viele Praxisbeispiele für die Umsetzung von mikrotypografischen Vorgaben gesammelt.

10.1 GREP-Abfragen automatisieren

In der Praxis ist es oftmals schneller, mehrere einfache GREP-Abfragen zu formulieren und hintereinander auszuführen, als lange an einer komplexen Abfrage zu feilen. Wenn die Aufgabe wiederholt ausgeführt werden soll, erweist es sich jedoch als sehr mühsam, diese immer wieder einzeln herauszusuchen und hintereinander auszuführen. Dann ist es besser, die Abfolge in einem Skript zu speichern.

GREP-Ketten bilden

Der erste und wichtigste Schritt ist, die einzelnen GREP-Abfragen im Suchen/Ersetzen-Dialog wie auf Seite 21 beschrieben zu speichern. Danach können Sie das Skript *ChainGREP.jsx* (Download unter http://www.publishingx.de/download/ChainGREP.jsx ↗113) verwenden, festgelegte Abfolgen aus den gespeicherten Abfragen bilden und diese GREP-Abfolgen in einem Skript speichern. Informationen zur Installation des Skripts finden Sie im Unterkapitel 2.1.

Nach dem Aufruf des Skripts aus der Skriptpalette erscheint ein Dialog, in dem alle GREP-Abfragen von InDesign angezeigt werden.

Die Liste kann komplett in einem Skript gespeichert werden. Meistens will man aber nur eine gewisse Auswahl an Abfragen hintereinander ablaufen lassen. Dazu können Sie mit Remove Abfragen aus der Auswahl entfernen und die Liste mit den Tasten Up bzw. Down sortieren. Nachdem Sie Ihre Auswahl getroffen haben, können Sie den Suchbereich der Abfragen im Bereich Scope of Find/Change festlegen. Nun müssen Sie noch einen Namen für das zu speichernde Skript eintragen. Die Skripte werden im Unterordner *FindChangeScripts* gespeichert. Mit der Auswahl Save List wird das Skript erstellt.

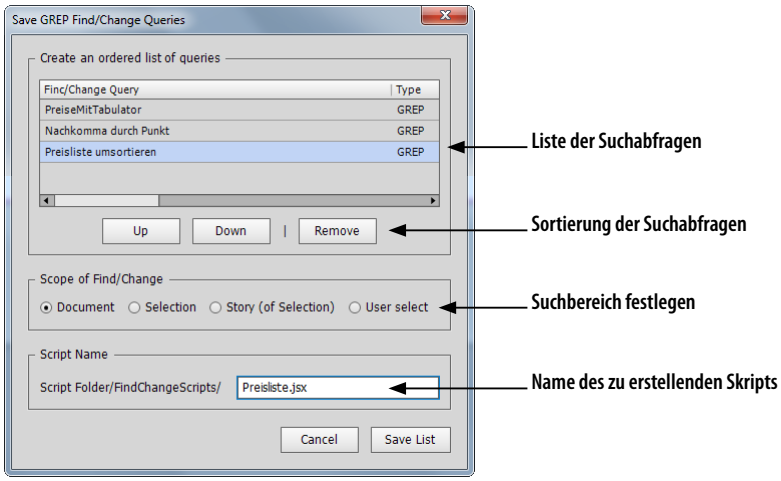


Abb. 84
Einstellungsmöglichkeiten des Skripts *ChainGREP.jsx*

Im Skript werden die Abfragen mit allen Einstellungen abgespeichert. Das heißt, Sie können das Skript auch an einem anderen Computer verwenden, an dem die Abfragen nicht installiert sind. Dies bedeutet aber auch, dass Sie bei einer Veränderung der ursprünglichen GREP-Abfrage das Skript neu erstellen müssen.

Eigenständiges Suchskript

Das Skript basiert auf dem *GREP query manager* 30 von Peter Kahrel, der noch mehr Einstellungsmöglichkeiten bietet.

10.2 Die besten GREP-Tools

Es gibt hilfreiche Tools und Informationen im Internet, die den Umgang mit Regulären Ausdrücken für InDesign erleichtern. Die wichtigste Quelle ist die eher unscheinbare Seite von Peter Kahrel 30, auf der Sie viele interessante Skripte finden.

Für komplexe Ausdrücke ist der *A GREP editor* zu empfehlen. Mit ihm können Abfragen in mehreren Zeilen – und damit deutlich übersichtlicher – eingegeben werden. Er enthält eine vollständige Liste von Platzhaltern und Metazeichen. Die InDesign-Suche kann direkt vom Skript-Dialog gesteuert werden.

GREP-Editor

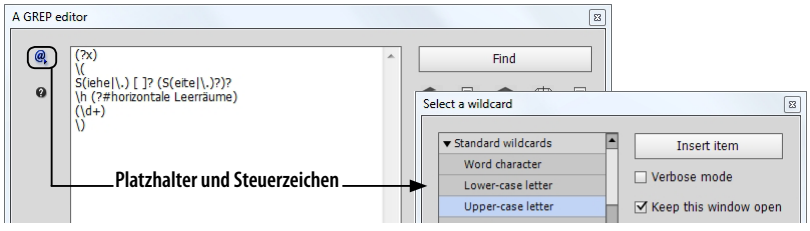


Abb. 85
GREP-Editor von Peter Kahrel

Sehr interessant ist auch die Sammlung *GREP mapper*, der die Idee zugrunde liegt, mit Hilfe von GREP-Stilen alle Zeichenklassen sichtbar zu machen. Die verschiedenen Unicode-Bereiche können mit einem Skript in einem bereits eingerichteten Template generiert werden.

GREP zuordnen

Neben den bekannten Standardzeichenklassen werden auch die Unicode-Zeichenklassen und die Posix-Zeichenklassen (→ Unterkapitel 10.8) unterstützt.

Peter Kahrel hat übrigens auch das Standardwerk *GREP in InDesign* verfasst, welches in englischer Sprache bei O'Reilly als E-Book erhältlich ist.

GREP-Fundstellen hervorheben

Mit dem Skript *highlightGrep.jsx* 158 von Roland Dreger kann man sich die Treffer einer GREP-Suche im Dokument anzeigen lassen. Im Live-Modus werden die Treffer schon während der Eingabe hervorgehoben. Dieses Skript empfiehlt sich besonders für Einsteiger.

What the GREP

Das Skript *What the GREP* von Theunis de Jong erklärt komplexe GREP-Ausdrücke. Dies ist besonders hilfreich, wenn Sie einen fertigen GREP haben, aber nicht so genau verstehen, wie er eigentlich funktioniert. Auf de Jongs Webseite 32 finden Sie außerdem eine Liste, welche Features von Regulären Ausdrücken in InDesign unterstützt werden.

tomaxxiGREP

Mit der InDesign-Erweiterung tomaxxiGREP 157 können GREP-Stile aktiviert und deaktiviert werden, ohne sie aus dem Absatzformat zu entfernen.

10.3 Grenzen und Übergänge

Im ersten Kapitel wurden bereits die Metazeichen `^` für den Beginn und `$` für das Ende des Absatzes vorgestellt. Wichtig bei diesen Markierungen ist, dass sie kein Zeichen, sondern eine Position markieren. Entsprechend wird mit `^` nicht das erste Zeichen eines Absatzes, sondern die Position vor dem ersten Zeichen gefunden – das erste Zeichen müsste man mit dem Ausdruck `^.` suchen. Dieses Verhalten wird gut sichtbar, wenn man eine Suche mit diesen Metazeichen ausführt: Es wird nichts markiert, nur die Einfügemarke springt an die entsprechende Stelle.

Mit dem Ausdruck `^\w+` findet man jeweils das erste Wort am Anfang eines Absatzes. Mit `[^.?!]$` findet man alle Absätze, die nicht mit einem Punkt, Frage- oder Ausrufezeichen enden.

Leere Textrahmen finden

Ganz ähnlich funktioniert das Metazeichen `\A`, das den Anfang eines Textabschnitts markiert, und entsprechend `\Z` für das Ende des Textabschnitts. Mit der Suche nach `\A\Z` finden Sie leere Textabschnitte, also einzelne Textrahmen ohne Inhalt.

Neben den offensichtlichen Textabschnitten bilden auch die Inhalte von Tabellenzellen jeweils einen eigenen Textabschnitt. Die Textabschnitte von Tabellenzellen werden allerdings erst für die Suche sichtbar, wenn sie mit Inhalt gefüllt sind. Das heißt, die Suche nach `\A\Z` findet keine leeren Tabellenzellen.

1 Schöner suchen und ersetzen mit GREP

Bei der Arbeit in InDesign kommt man oft mit der Suchen/Ersetzen-Funktion in Berührung. Viele Anwender beschränken sich auf die Suche nach bestimmten Texten, die gegebenenfalls durch andere Texte ersetzt werden. Dies ist allerdings nur das kleinste und einfachste Werkzeug aus dem großen Suchen/Ersetzen-Werkzeugkasten.

Reguläre Ausdrücke

InDesign bietet ab der Version CS3 neben der normalen Suchfunktion die Suche mit *Regulären Ausdrücken* bzw. GREP*. Hier haben Sie die Möglichkeit, nach Textmustern (z. B. Zahlen, Leerräumen, Großbuchstaben etc.) zu suchen. Ein Beispiel ist die Suche nach allen Zahlen von 0 bis 9 mit einem einzigen Suchbefehl. Die gefundenen Zeichen können dann bei der Ersetzung wiederverwendet werden. Stellen Sie sich vor, Sie müssten alle Seitenverweise in einem Dokument von der Form (`S. 100`) in die Form → `Seite 100` bringen. Mit GREP brauchen Sie dafür nur eine Ersetzungsanweisung, weil die unterschiedlichen Seitenzahlen in der Ersetzung wieder eingesetzt werden.

Seit InDesign CS4 gibt es zusätzlich die Möglichkeit, in Absatzformaten so genannte GREP-Stile festzulegen. Hier werden bestimmte Zeichenfolgen definiert, die besonders formatiert werden sollen. Typisch wäre die Kleinerschaltung von Ziffern im Buchsatz oder die Hervorhebung von Namen. In InDesign CS6 wurden zwei Befehle hinzugefügt, die bei der Suche nach Leerräumen und Absätzen hilfreich sind.

Auf den ersten Blick sehen die Suchanfragen unübersichtlich und kompliziert aus. Dies ist der Abstraktion geschuldet, die gleichzeitig erst die Mächtigkeit der Suchausdrücke ermöglicht. Solange die Anfragen präzise und logisch formulierbar sind, kann man praktisch alles finden.

Reguläre Ausdrücke sind mit einigen Grundlagen schnell zu erlernen. Für den produktiven Einsatz ist es auch gar nicht notwendig, die letzten Details und Kniffe zu kennen. Statt theoretisch Möglichkeiten zu diskutieren, ist es oft hilfreicher, das grundsätzliche Prinzip zu verstehen und bei schwierigen Anfragen zu wissen, wie man weiterkommt.

---

* Die genaue Bedeutung von GREP ist umstritten, ich finde **G**eneral **R**egular **E**xpression **P**arser passend. Wer es genau wissen will, kann unter ↗ 122 nachsehen.

Im Folgenden stelle ich typische Suchanfragen vor, die man am besten mit GREP lösen kann. Im weiteren Verlauf des Buches werden noch komplexere Such- bzw. Ersetzungsanweisungen (→ Kapitel 10) und Möglichkeiten zum Skripten von GREP-Anfragen besprochen (→ Unterkapitel 4.8).

1.1 Der Suchen/Ersetzen-Dialog

Wenn Sie mit Befehl + F den Suchen/Ersetzen-Dialog aufrufen, sehen Sie die vier Registerkarten für die verschiedenen Suchtypen und die Bedienelemente für die Ausführung der Anweisungen. Der Dialog ist übrigens nicht modal, d.h., Sie können zwischen Dokument und Dialog hin und her wechseln und z.B. per Copy & Paste Text, Leerräume und Sonderzeichen aus dem Dokument übernehmen.

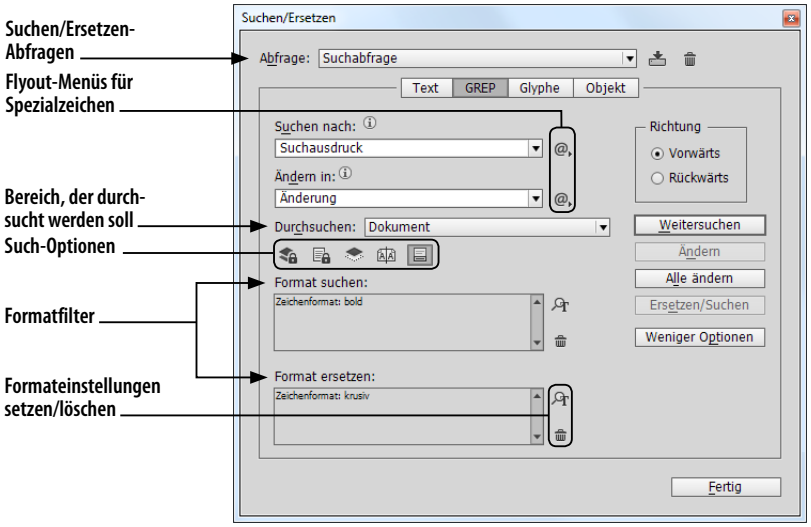


Abb. 1
Suchen/Ersetzen-Dialog

Mit den Registerkarten Text und GREP können Texte durchsucht und Ersetzungen vorgenommen werden. Der Aufbau und die Such-Optionen der beiden sind nahezu gleich. In der Registerkarte GREP hat man die Möglichkeit, mit Regulären Ausdrücken zu arbeiten.

Mit der Registerkarte Glyphe können bestimmte Zeichen anhand ihrer ID, mit der Registerkarte Objekt Rahmen anhand von Attributen gesucht werden. Diese Registerkarten werden in diesem Buch nicht vorgestellt, weil man sie in der Praxis nur selten benötigt.

Die Registerkarte GREP

Die Registerkarte GREP enthält die Texteingabefelder Suchen nach und Ändern in, die Buttons zur Einstellung der Such-Optionen und die Felder für die Formateinstellungen.

Im Suchen- bzw. Ändern-Feld werden die Regulären Ausdrücke für Suchanfragen bzw. Ersetzungsanweisungen eingetragen. Für die Arbeit

mit Regulären Ausdrücken werden einige Platzhalter und Zeichen mit einer speziellen Bedeutung benötigt. Diese finden Sie neben den Feldern im Flyout-Menü des @-Buttons.

Unter den Texteingabefeldern kann mit einer Dropdown-Liste der Bereich, der durchsucht werden soll, festgelegt werden. Im Normalfall finden Sie hier die Auswahl zwischen allen Dokumenten und dem aktiven Dokument. Wenn sich die Einfügemarke beim Aufruf des Dialogs in einem Text befindet, finden Sie zusätzlich die Möglichkeiten, den aktuellen Textabschnitt oder den Textabschnitt von der Einfügemarke bis zum Ende zu durchsuchen. Der Textabschnitt beinhaltet den Text des Textrahmens oder den Text der miteinander verketteten Rahmen. Falls Sie einen Text ausgewählt haben, gibt es noch die Möglichkeit, die Auswahl zu durchsuchen.

Such-Optionen

Unterhalb der Dropdown-Liste können die Such-Optionen eingestellt werden. Mit ihnen kann festgelegt werden, welche Bereiche durchsucht werden sollen. Im Normalfall ist nur die Suche innerhalb von Fußnoten eingeschaltet. Weiterhin kann man für die Suche und Ersetzung auch die Musterseiten und ausgeblendete Ebenen miteinbeziehen. Gesperrte Ebenen bzw. Objekte und gesperrte Textabschnitte können nur bei der Suche berücksichtigt werden. Zu den Such-Optionen zählt auch die ab InDesign CC vorhandene Möglichkeit, Vorwärts bzw. Rückwärts zu suchen.

Im Gegensatz zu der Textsuche sind in der Registerkarte GREP keine Buttons für die Suche nach einem ganzen Wort und für die Beachtung der Groß-/Kleinschreibung vorhanden. Diese Einstellungen werden bei Regulären Ausdrücken im Suchausdruck festgelegt.

Die Suchen/Ersetzen-Funktion von InDesign geht weit über das Suchen und Ersetzen von Buchstaben, Wörtern oder Satzteilen hinaus. InDesign bietet die Möglichkeit, die Suche mit der Angabe von Textformatierungen und Absatzattributen weiter zu verfeinern. Bei der Ersetzung können alle diese Formateinstellungen dann geändert bzw. gesetzt werden. Diese Einstellungen werden in den Feldern Format suchen bzw. Format ersetzen angezeigt. Mit der Lupe gelangen Sie zur Formatauswahl, mit dem Mülleimer können Sie die Formateinschränkungen wieder entfernen.

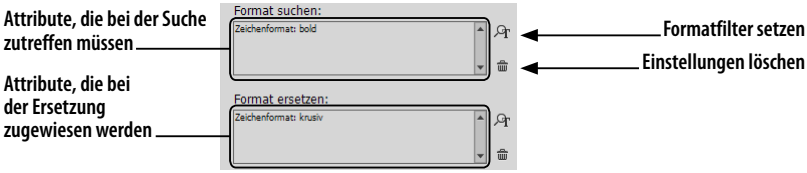


Abb. 2
Fette Textstellen kursiv formatieren

Vermutlich ist Ihnen auch schon aufgefallen, dass der Dialog alle Einstellungen nach dem Schließen beibehält. InDesign vergisst die Einstellungen erst beim Neustart des Programms.

Suchen/Ersetzen-Abfragen abspeichern

Es können auch komplette Suchen/Ersetzen-Abfragen abgespeichert werden. Dazu finden Sie im Dialog eine Dropdown-Liste und den Button 📥 zum Speichern bzw. 🗑 Löschen der aktuellen Abfrage. Interessante Abfragen kann man so für die spätere Verwendung speichern. Beim Skripting kann man auf die so gespeicherten Abfragen wieder zugreifen. Mit Hilfe des Skripts *ChainGREP.jsx* können diese Abfragen verkettet und gesammelt ausgeführt werden (→ Unterkapitel 10.1).

1.2 Die Suche mit Regulären Ausdrücken

Zum Einstieg empfiehlt es sich, einfach Suchanfragen auszuprobieren. Wenn Sie alle Vorkommen in einem Dokument ersetzen, sollten Sie allerdings Vorsicht walten lassen, da die Gefahr besteht, dass ungewollte Ersetzungen durchgeführt werden.

Zur besseren Übersichtlichkeit sind alle Suchanfragen und Ersetzungsanweisungen in diesem Kapitel `grau` hinterlegt.

Das Einfachste vorab: Ganz normale Suchtexte funktionieren auch mit GREP. Wenn nach `Max` gesucht wird, kann der Name durch `Moritz` ersetzt werden.

1.2.1 Zeichen mit spezieller Bedeutung

Mit Regulären Ausdrücken kann man variable Zeichenketten suchen und Suchergebnisse präzise eingrenzen. Diese Flexibilität wird dadurch erreicht, dass einige Zeichen eine besondere Bedeutung haben.

Metazeichen

Wenn man z. B. im Texteingabefeld Suchen `~S` eingibt, findet InDesign geschützte Leerzeichen ^ und nicht die beiden Zeichen ~ und S. Hier wird ein Sonderzeichen durch die Kombination aus dem Tilde-Zeichen mit einem anderen Buchstaben dargestellt. Das Tilde-Zeichen ist hier ein *Metazeichen*. Metazeichen stehen nicht für sich selbst, sondern haben eine besondere Bedeutung.

Maskierung

Neben dem Tilde-Zeichen gibt es noch einige weitere Zeichen, die eine spezielle Bedeutung haben. So findet man z. B. mit dem Punkt `.` nicht nur das Satzendzeichen, sondern fast jedes beliebige Zeichen. Wenn nur nach einem Punkt gesucht wird, muss die besondere Bedeutung aufgehoben werden. Dies geschieht mit einem vorangestellten Backslash. Nach einem Punkt muss also mit `\.` gesucht werden. Diese Aufhebung der Bedeutung von Zeichen mit spezieller Bedeutung nennt man *Maskierung* (engl. escape) – sie wird übrigens ganz ähnlich auch beim Skripting benötigt.

In Regulären Ausdrücken gibt es Zeichen, um Sonderzeichen zu erzeugen, die Suche genauer zu steuern oder etwas im Suchausdruck zu markieren. Diese Zeichen müssen alle mit dem Backslash maskiert werden, wenn sie buchstäblich gefunden werden sollen. Der Backslash zählt auch dazu und wird mit `\\` gefunden.

Metazeichen bei der Suche mit GREP

```
\ . * + ? ( ) { } [ ^ $ | ~
```

In InDesign CS6 funktioniert \$ nicht. Verwenden Sie [$]

Wenn man `wo bist du?` finden möchte, muss man nach `wo bist du\?` suchen. Zunächst ist es nur wichtig, die Metazeichen zu kennen. Sie werden im Verlauf des Kapitels vorgestellt.

Die Idee, die Suchanfragen mit Metazeichen zu verfeinern, gab es auch schon vor InDesign CS3 in der normalen Textsuche. Dort gibt es z. B. die Möglichkeit, mit `^p` nach einem Absatzende oder mit `^?` nach einem beliebigen Zeichen zu suchen. Hier wird das Caret-Zeichen (Zirkumflex) ^ zur Kennzeichnung von Zeichen mit spezieller Bedeutung verwendet. Wenn man es finden möchte, muss man es ebenfalls mit `^^` maskieren. Diese Möglichkeiten sind für die Textsuche in der Registerkarte Text erhalten geblieben.

Die Problematik mit den Metazeichen ist der einzige Grund, zuweilen noch die normale Suche in der Registerkarte Text zu verwenden. Wenn man nur eine bestimmte Zeichenfolge sucht, muss man sich dort (mit Ausnahme des Caret-Zeichens) nicht um die Maskierung kümmern.

1.2.2 Variable Zeichen auswählen

Platzhalter

Die wichtigsten GREP-Befehle sind die so genannten *Platzhalter* (engl. wildcard), mit denen unterschiedliche Zeichen gefunden werden. Sie ermöglichen eine unscharfe Suche nach Zeichengruppen wie allen Kleinbuchstaben oder allen Ziffern.

Die im Folgenden vorgestellten Zeichen mit einer speziellen Bedeutung finden Sie im Flyout-Menü @, neben dem Suchen/Ersetzen-Textfeld.

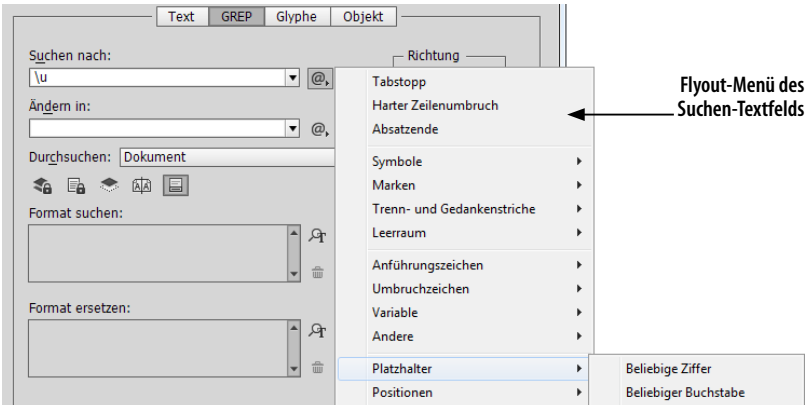


Abb. 3 Platzhalter Flyout-Menü

Beliebige Zeichen finden

Der wichtigste und einfachste Platzhalter ist der Punkt `.`. Mit ihm kann jedes beliebige Zeichen, mit Ausnahme vom Absatzende (hierzu zählen der normale Zeilenumbruch ¶ und auch alle anderen Umbruchzeichen wie z. B. Seitenumbruch) und dem harten Zeilenumbruch (Soft-Return) ¬, gefunden werden. Mit `..` würde man zwei beliebige aufeinanderfolgende Zeichen finden. Platzhalter repräsentieren eine bestimmte Menge an Zeichen und werden deswegen als *Zeichenklassen* bezeichnet.

Die Suche nach `Max.` findet nicht nur `Max.`, sondern auch `Maxi` mit i und ohne Punkt sowie alle anderen Formen von `MaxO` bis `Max:`. Mit der Suche nach `d..` könnte man z. B. alle Wörter finden, die mit `d` beginnen und mindestens drei Buchstaben lang sind.

Ein Platzhalter trifft genau ein Zeichen. Wenn Sie einen Punkt in das Suchen-Feld eingeben und die Suche mehrmals ausführen, wird ein Zeichen nach dem anderen gefunden. Um die Verwendung von Zeichenklassen zu verdeutlichen, greife ich auf das nächste Unterkapitel vor und führe die Möglichkeit ein, mit dem Plus-Zeichen mehrere Zeichen vom gleichen Typ zu suchen. Wenn man ein `+` hinter eine Zeichenklasse setzt, wird das erste und alle folgenden zusammenhängenden Zeichen der Zeichenklasse gefunden. Wenn man das Plus-Zeichen mit der Zeichenklasse Punkt kombiniert `.+`, werden entsprechend alle Zeichen bis zum Absatzende oder einem harten Zeilenumbruch gefunden.

Zeichenklassen

Neben dem Punkt gibt es noch einige andere nützliche Zeichenklassen. Diese werden alle aus Kombinationen mit dem Backslash gebildet.

Mit `\u` findet man einen beliebigen Großbuchstaben, mit `\l` einen Kleinbuchstaben. Mit `M\l+` findet man entsprechend alle Wörter, die mit einem großen `M` beginnen und denen Kleinbuchstaben folgen: `Max`, `Moritz`, `Montag`, `Mars` usw., aber nicht `MacOS`.

Entsprechend findet man mit `\u\l+` alle Wörter, die mit einem Großbuchstaben beginnen, mit `\l+` alle kleingeschriebenen.

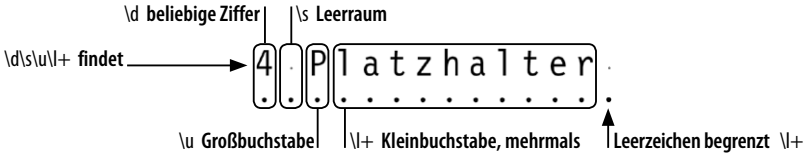


Abb. 4
Verhalten von Platzhaltern

Eine weitere wichtige Zeichenklasse sind Zahlen. Alle Ziffern von 0 bis 9 können mit `\d` gefunden werden. Alle beliebigen Ziffern in einem Text finden Sie mit `\d+`. Später zeige ich noch die Möglichkeit, auch Ziffern mit Nachkommastellen in den Suchausdruck miteinzubeziehen.

Mit `\s` findet man alle Leerräume, also u. a. Leerzeichen, geschützte Leerzeichen ^, Festabstände wie z. B. Halbgeviert ⁻, den Tabulator » und Zeilenschaltungen ¶ bzw. ¬.

Die Platzhalter `\h` und `\v` wurden in InDesign CS6 eingeführt.

Mit InDesign CS6 hat Adobe die Platzhalter `\h` für alle horizontalen Leerräume (Leerzeichen, Tabulator etc.) und `\v` für alle vertikalen Leerräume (Absatzende und harter Zeilenumbruch) eingeführt. Bei anspruchsvollen Layouts, in denen mikrotypografische Anpassungen vorgenommen werden sollen, kann man mit `\h` sehr gut verschiedene Kombinationen suchen und durch eine Ersetzung vereinheitlichen. Mit dem Suchausdruck `\d\hmm` findet man z. B. alle Kombinationen einer

Zahl und der angehangenen Maßeinheit `mm` unabhängig vom Leerraum, der dazwischen verwendet wurde.

Zeichenklassen negieren

Außerdem hat man die Möglichkeit, die Bedeutung der Zeichenklassen umzudrehen, d.h. also alle Zeichen, die ihr *nicht* entsprechen, zu suchen. So kann man mit `\D` alles außer Ziffern finden – hier findet man neben Buchstaben auch Leerräume, Satzzeichen etc.

Tab. 1 Oft verwendete vordefinierte Zeichenklassen

Zeichenklasse	Beschreibung
`.`	Ein beliebiges Zeichen, aber nicht das Absatzende (Return) oder einen erzwungenen Zeilenumbruch (Soft-Return)
`\u`	Alle Großbuchstaben (engl. uppercase). Findet keine elektronisch versal TT gestellten Texte.
`\l`	Alle Kleinbuchstaben (engl. lowercase)
`\d`	Alle Ziffern (engl. digits) 0–9. Keine Brüche und nur elektronisch hoch- oder tiefgestellte Zahlen.
`\s`	Alle Leerräume, also Leerzeichen, alle Festabstände, Tabulatoren und Umbruchzeichen (engl. space). Fälschlicherweise auch das selten verwendete »Einzug bis hierhin«- Zeichen †.
`\h`	Alle horizontalen Leerräume: Leerzeichen, alle Festabstände und den normalen Tabulator (nicht den Tabulator für rechte Ausrichtung).

Zeichenauswahl

Im Flyout-Menü finden Sie die Zeichenauswahl unter Entsprechung → Zeichensatz

Etwas genauer als mit Zeichenklassen kann man mit einer *Zeichenauswahl* suchen. Hier können die Zeichen, die gesucht werden sollen, selbst festgelegt werden – man kann sozusagen seine eigene Zeichenklasse definieren. Eine Zeichenauswahl wird innerhalb von eckigen Klammern `[]` festgelegt. So findet man mit dem Ausdruck `S[kc]ript` die Variationen `Skript` und `Script`. Innerhalb der eckigen Klammern können beliebig viele Zeichen eingetragen werden, die im Suchtreffer an der entsprechenden Position vorkommen können. Ein anderes Beispiel wäre die Suche nach `Buch[st][st]abe`, mit der man z.B. Buchstabendreher zwischen `s` und `t` auflösen kann. Mit der Suche nach `[gjpqy]` wiederum würde man alle Buchstaben mit Unterlängen finden – bei kursiven Schriften kämen noch *f* und *ß* hinzu.

Noch flexibler wird eine Zeichenauswahl durch die Möglichkeit, Bereiche festzulegen. Bereiche werden mit einem Bindestrich zwischen dem ersten und letzten Zeichen des Bereichs definiert. Die Anfrage `200[5-9]` findet alle Jahreszahlen zwischen `2005` und `2009`. Mit der Anfrage `[A-N]` findet man alle Großbuchstaben zwischen `A` und `N`. Solche Bereiche kann man gut zur Formatierung von Listen, in denen z.B. Produktnummern nach einem bestimmten Schema festgelegt sind, verwenden. Vorstellbar ist z.B. eine Produktnummer, die mit `1` oder `2` beginnt, dann mit einem Buchstaben zwischen `A` und `N` weitergeht, gefolgt von noch zwei weiteren Ziffern. Der passende Ausdruck dafür wäre `[12][A-N]\d\d`.

Negative Zeichenauswahl

Weiterhin können auch negative Zeichenmengen angegeben werden. Hier kann man Zeichen festlegen, die nicht vorkommen dürfen. Dies wird mit dem Zirkumflex bzw. Caret-Zeichen ^ erreicht. Die Anfrage `InDesign CS[^3-9]` findet alle InDesign CS-Versionen vor CS3, wobei die Version CS auch ohne die Ziffer 1 gefunden wird, da das folgende Zeichen – vermutlich ein Leerzeichen – ebenfalls nicht in der Menge aller Zeichen zwischen `3` und `9` enthalten ist. Genauso könnte man mit `Abbildung\h[^\d]` alle Abbildungslegenden finden, die noch keine Nummerierung haben.

Innerhalb der Zeichenauswahl gibt es andere Metazeichen, hier haben nur die schließende eckige Klammer], der Zirkumflex ^, das Tilde-Zeichen ~ und der Bindestrich (Divis) – eine besondere Bedeutung. Für die Maskierung braucht man wieder den Backslash. Zwischen den eckigen Klammern sind nur die folgenden Zeichen reserviert:

```
] ^ - \ ~
```

Metazeichen bei der Festlegung einer Zeichenauswahl

Die im vorhergehenden Abschnitt genannten Metazeichen haben hier keine besondere Bedeutung mehr und müssen nicht maskiert werden. Die Suche nach `[.\~]` findet Punkte und Tilde-Zeichen. Die Maskierung des Bindestrichs kann am Ende der Zeichenauswahl entfallen `[.\~-]`.

Innerhalb einer Zeichenauswahl können auch Zeichenklassen eingesetzt werden. Mit `[\d,]` findet man Zahlen mit Nachkommastellen – eventuell muss noch der Punkt hinzugefügt werden.

Mit `[\dA-N]+` würde man wieder die oben erwähnte Produktnummer finden. Diese Anfrage ist aber weit weniger präzise, weil Reihenfolge und Länge der Nummer nicht festgelegt sind.

Wörter finden

Mit dem Ausdruck `[\u\l\-]+` könnte man die Zeichenklassen von Groß- und Kleinbuchstaben sowie den Bindestrich suchen, also praktisch alle Wörter. Vielleicht kennen oder finden Sie die Zeichenklasse `\w`, die »offiziell« für diesen Zweck gedacht ist. Mit `\w` findet man jedoch alle Buchstaben, Zahlen und den Unterstrich, für die Suche nach Wörtern ist das eher unbrauchbar.

E-Mail-Adressen finden

Als weitere praktische Anwendung stelle ich die Suche nach einer E-Mail-Adresse vor. Dreh- und Angelpunkt ist das @-Zeichen. Vor dem @ dürfen sich Buchstaben, Zahlen und Unterstriche befinden, die mit `\w` gefunden werden können. Da aber auch Punkte und Bindestriche erlaubt sind, muss eine eigene Zeichenklasse gebildet werden `[\w.-]+`. Nach dem @-Zeichen sind die gleichen Zeichen erlaubt. Zur genaueren Eingrenzung sollte die Domainendung, die sich hinter einem Punkt befindet und nur noch aus den Zeichen a–z bestehen darf, angefügt werden `\.[a-z]+`. Zusammengefügt ergibt sich der Ausdruck `[\w.-]+@[\w.-]+\.[a-z]+`. Bitte beachten Sie, dass E-Mail-Adressen sehr komplex sein können und mit der hier vorgestellten GREP-Suche nur die üblichen Varianten gefunden werden.

Alternativen

Wenn man nicht nach Zeichenmengen, sondern nach variierenden Wortteilen suchen möchte, müssen die Variationen in runde Klammern geschrieben und mit dem senkrechten Strich | unterteilt werden.

Die Suche nach `(Java|Apple|VB)Script` findet alle drei Programmiersprachen innerhalb des Textes. Genauso kann natürlich auch einfach nach zwei alternativen Wörtern gesucht werden – `(Nela Malou|Silke)` findet beide Vorkommen im Text. Es liegt auf der Hand, dass diese Suchanfrage deutlich präziser ist als die ebenfalls erfolgreiche Suche nach allen Buchstaben der beiden Namen mit `[NMSelaouik]+`.

1.2.3 Sonderzeichen

Viele Zeichen können nicht direkt in die Such- bzw. Ersetzungsanfrage eingegeben werden. So ist es z. B. nicht möglich, eine Zeilenschaltung direkt in das Texteingabefeld einzugeben. Bei anderen Zeichen ist dies zwar prinzipiell möglich, aber ein geschütztes Leerzeichen wäre im Texteingabefeld nicht von einem normalen Leerzeichen zu unterscheiden. Die dritte Klasse bilden die Zeichen mit einer besonderen Bedeutung. Wenn die eigentlichen Zeichen gefunden werden sollen, müssen diese, wie am Anfang beschrieben, mit dem Backslash maskiert werden.

Tabulator und Zeilenumbrüche werden in Regulären Ausdrücken mit dem Backslash erzeugt. Einen oder mehrere Tabulatoren findet man z. B. mit dem Suchausdruck `\t+`.

Tab. 2 Standardsonderzeichen für die Suche mit Regulären Ausdrücken

GREP	Zeichen	Beschreibung
\t	»	Tabulator
\r	¶, ⁖, ❡ …	Absatzende bzw. Zeilenschaltung (Return), aber auch Rahmenumbruch, Seitenumbruch etc.
\n	¬	Erzwungener Zeilenwechsel (Soft-Return)

Alle weiteren Sonderzeichen werden mit der Tilde ~ erzeugt. Alle Sonderzeichen, die so erzeugt werden, finden Sie nach Bereichen sortiert im zweiten und dritten Bereich des Flyout-Menüs neben dem Suchen-Feld.

Abb. 5 Sonderzeichen-Auswahl im Suchen/Ersetzen-Dialog

Nach einem geschützten Leerzeichen sucht man z.B. mit `~S`, nach bedingten Trennzeichen mit `~-`. Interessant ist die Möglichkeit, dass auch Indexmarken `~I`, verankerte Objekte `~a` und Textvariablen `~v` gesucht werden können.

Grundsätzlich können Sie alle Zeichen auch per Copy & Paste in die Texteingabefelder des Suchen/Ersetzen-Dialogs einfügen. InDesign erzeugt dann automatisch die richtige Kombination mit der Tilde. Die folgende Tabelle zeigt einige oft gebräuchliche Sonderzeichen, eine vollständige Übersicht finden Sie im Anhang A1.

Tab. 3 InDesign-Sonderzeichen in Regulären Ausdrücken

GREP	Zeichen	Beschreibung
~S	^	Geschütztes Leerzeichen
~m	—	Geviert
~>	–	Halbgeviert
~<	˙	Achtelgeviert
~b	¶	Standardzeilenumbruch
~=	–	Halbgeviertstrich, Gedankenstrich
~-	-	Bedingter Trennstrich
~y	⇥	Tabulator rechte Ausrichtung
~I	⁝	Indexmarke
~a	¥	Marker für verankerte Objekte
~v		Alle Textvariablen
\~	~	Tilde

Kombinationen mit der Tilde gelten nur in InDesign und können nicht in anderen Programmen, die Reguläre Ausdrücke unterstützen, eingesetzt werden.

Sonderzeichen und Zeichenklassen können in einem alternativen Suchausdruck oder einer eigenen Zeichenklasse kombiniert werden. So könnte man der Zeichenklasse `\h` mit dem Ausdruck `(\h|~y)` bzw. `[\h~y]` den fehlenden Tabulator rechte Ausrichtung hinzufügen.

Unicode-Zeichen finden

Neben diesen festgelegten Zeichen gibt es auch die Möglichkeit, ein Zeichen über den *Unicode*-Codepoint festzulegen. Dazu muss der hexadezimale Codepoint des Zeichens bekannt sein. Nach dem Zeichen ¾ suchen Sie mit `\x{00BE}`, das €-Symbol finden Sie mit `\x{20AC}`. Den Codepoint eines markierten Zeichens kann man in InDesign in der Mitte des Informationsfensters ablesen (Fenster → Informationen). Weitere Informationen zu Unicode finden Sie auf Seite 148. Im Kapitel 10.7 werden weitere Suchtechniken vorgestellt.

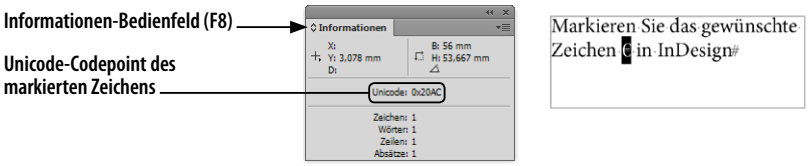


Abb. 6 Informationen-Bedienfeld

Wenn der Codepoint dort zu sehen ist, können Sie das Zeichen aber genauso gut per Copy & Paste in die Texteingabefelder des Dialogs befördern. Die Definition eines Zeichens mit `\x{####}` benötigt man nur, wenn ein Unicode-Zeichen gesucht wird, dessen Codepoint aus einer Tabelle oder einem anderen Programm bekannt ist.

1.2.4 Wiederholungen

Eine weitere wichtige Technik in Regulären Ausdrücken ist es, die Anzahl von Wiederholungen eines Zeichens oder einer Zeichenklasse festzulegen. Das Wiederholungszeichen `+`, mit dem man mehrere Zeichen finden kann, wurde bereits verwendet.

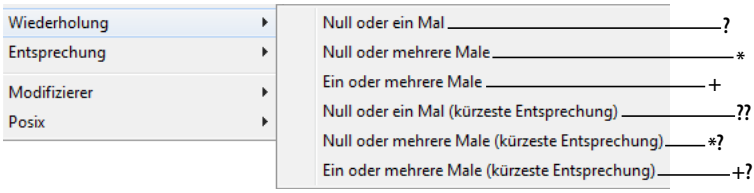


Abb. 7 Wiederholung Flyout-Menü

Wiederholungszeichen

Immer wenn man nicht genau weiß, ob ein Suchbegriff oder Zeichen im Text vorkommt, kann man das Wiederholungszeichen `?` verwenden (im Menü NULL ODER EIN MAL). So könnte man z.B. alle Vorkommen von `das` und `dass` mit der Suchanfrage `dass?` prüfen. Mit `(Max)? Mustermann` finden Sie jede Erwähnung von Herrn Mustermann – mit und ohne Vorname. Das Wiederholungszeichen, auch *Quantifizierer* genannt, bezieht sich immer auf das vorhergehende Zeichen, eine Zeichenklasse oder das vorhergehende Wort in runden Klammern. Es kann nicht alleine stehen.

Mit einem `+` wird festgelegt, dass ein Suchausdruck mindestens einmal vorkommen muss, aber beliebig oft vorkommen darf (im Menü EIN ODER MEHRERE MALE). Wenn Sie z.B. mehrere Leerzeichen hintereinander zu einem einzigen vereinheitlichen wollen, kann man mit `+` eine beliebige Anzahl von Leerzeichen finden. Verwenden Sie dazu nicht `\s+`, da in der Zeichenklasse auch die Absatzschaltungen enthalten sind und Sie eventuell die Kombination Leerzeichen Return entfernen würden.

Ein Plus bietet sich oft an, um den Rest eines Absatzes oder den variablen Text zwischen zwei bekannten Zeichenfolgen zu finden. Wenn ein beliebiger Text, z.B. die Beschreibung nach einer Produktnummer

oder vor dem Preis, gesucht wird, helfen die speziellen Zeichenklassen meist nicht weiter, da die Beschreibung fast alle Zeichen enthalten kann. Angenommen einer wie oben geformten Produktnummer folgt eine Produktbeschreibung, z. B. `1G78 Staubsauger`. Die Nummer inklusive Beschreibung finden Sie jetzt mit `[12][A-N]\d\d\h.+` – es wird sogar `1G79 Staubsauger mit Beutel` gefunden oder `1H32 Wischmops`.

Wenn man nicht genau weiß, ob das Zeichen oder die Zeichenklasse überhaupt vorkommt, kann mit einem `*` auch nach beliebig vielen Vorkommen gesucht werden (im Menü Null oder mehrere Male, korrekt wäre die Bezeichnung Beliebig oft). Der Suchbegriff kann keinmal, einmal oder beliebig oft vorkommen. Im Unterschied zum `?` findet man mit dem `*` eine beliebige Anzahl von Zeichen – mit `jaa*` findet man `ja`, `jaa`, `jaaa` usw. Dies ist z. B. nötig, um nach einer Zahl zu suchen, die eventuell Nachkommastellen haben kann. Der Suchausdruck `\d+,?\d*` findet sowohl `15` als auch `14,9999`, wobei die Anzahl der Nachkommastellen nicht relevant ist. Im Suchausdruck könnte man auch das Komma mit einem `*` versehen. Dies wäre aber nicht besonders präzise, weil bei einer Zahl mit Nachkommastellen genau ein, bei einer Zahl ohne Nachkommastellen genau kein Komma enthalten ist.

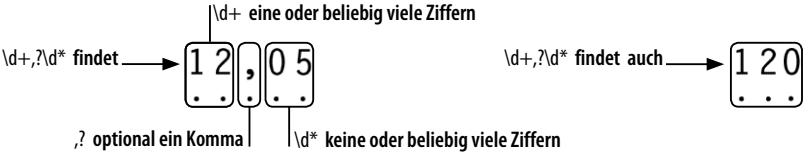


Abb. 8
Wiederholungen

Achten Sie darauf, dass das Sternchen in Kombination mit nur einem Zeichen immer trifft – eine solche Suche ist also nicht sinnvoll. Beispielsweise findet `x*` an jeder Position des Textes entweder ein oder kein `x`.

Wiederholungszeichen	Beschreibung
?	Das Zeichen oder die Zeichenklasse ist optional, d. h., sie kann einmal oder keinmal vorkommen.
+	Der Suchbegriff muss mindestens einmal, kann aber beliebig oft vorkommen.
*	Der Suchbegriff kann beliebig oft vorkommen – also auch keinmal.

Tab. 4
Wiederholungsfaktoren in Regulären Ausdrücken

Suchverhalten der Wiederholungszeichen

Die Wiederholungszeichen `+` und `*` finden immer so viele Zeichen wie möglich. Sie suchen so lange weiter, bis sie den größtmöglichen passenden Suchausdruck gefunden haben. Das ist auch der Grund, warum man mit `.+\.` nicht den ersten Satz eines Absatzes auswählen kann, wie man vielleicht vermuten könnte. In diesem Ausdruck sammelt das `+` beliebige Zeichen vom Absatzbeginn bis zum letzten Vorkommen eines

InDesign versucht normalerweise den längsten Treffer zu ermitteln.

Punkts und damit meist den ganzen Absatz. Dieses Verhalten wird als *gierig* (engl. greedy) bezeichnet.

Das Fragezeichen hat je nach Kontext unterschiedliche Bedeutungen.

Um die Wiederholungszeichen zu zügeln, kann ein Fragezeichen verwendet werden. Hinter einem Wiederholungszeichen bedeutet es, dass nur die kleinstmögliche Anzahl Treffer gefunden werden soll. Mit dem Ausdruck `.+?\.` wird nur noch der erste Satz eines Absatzes gefunden. Das Fragezeichen hat also mehrere Bedeutungen: Hinter einem Zeichen oder einer Zeichenklasse bewirkt es, dass das Suchzeichen im Treffer ein- oder keinmal vorkommen darf. Hinter einem Wiederholungszeichen regelt es, dass so wenige Zeichen wie möglich gefunden werden.

Das Fragezeichen sollte immer eingesetzt werden, wenn die Gefahr besteht, dass die nächste Fundstelle nach einem Wiederholungszeichen nicht eindeutig ist. Wenn z. B. in Klammern gesetzter Text gesucht wird (`also dieser hier`), ist es durchaus möglich, dass innerhalb des Absatzes noch ein weiterer Text in runden Klammern steht (`hier ist der zweite`). Wenn man nun mit einem gierigen Wiederholungszeichen `\(.+\)` sucht, findet man den gesamten Text von der ersten bis zur letzten runden Klammer. Hier sollte man stattdessen mit dem Ausdruck `\(.+?\)` arbeiten. Achten Sie auch darauf, dass die spezielle Bedeutung der Klammern durch die Maskierung mit dem Backslash aufgehoben werden muss.

Genaue Anzahl der Vorkommen festlegen

Noch genauer kann mit geschweiften Klammern `{}` festgelegt werden, wie oft ein Zeichen oder eine Zeichenklasse vorkommen darf. Eine Zahl innerhalb der geschweiften Klammern legt fest, wie viele Vorkommen des Zeichens oder der Zeichenklasse erlaubt sind. So findet man mit `\d{4}` alle vierstelligen Zahlen.

Wenn man einen Wiederholungsbereich festlegen möchte, muss die minimale und maximale Anzahl der Vorkommen angegeben werden. Beide Werte werden durch ein Komma getrennt. Zwei- oder dreistellige Zahlen findet man entsprechend mit `\d{2,3}`.

Tab. 5 Genaue Anzahl von Wiederholungen festlegen

Wiederholungen	Beschreibung
`{n}`	Der Ausdruck muss exakt n-mal vorkommen.
`{n,m}`	Der Ausdruck muss mindestens n-mal und darf maximal m-mal vorkommen.
`{n,}`	Der Ausdruck muss mindestens n-mal und darf öfter vorkommen.

1.2.5 Genaue Positionen ermitteln

Suchbereich

Eine Suche kann oft durch Positionsangaben im Suchbereich noch genauer eingegrenzt werden. Standardmäßig endet der Suchbereich nach dem Absatzende oder dem harten Zeilenumbruch – dem ersten Zeichen, das nicht von `.` gefunden wird.

Auf welcher Seite bin ich?

Die Auswahl des Textrahmens mit dem Namen `seitenzahl` in Zeile 14 des letzten Skripts ist auf die linke Musterseite beschränkt. Falls auch mit dem Textrahmen auf der rechten Seite gearbeitet werden soll, benötigt man eine Möglichkeit, mit der entweder die rechte oder linke Musterseite ausgewählt werden kann. Wie die Position der Seite relativ zum Bund ermittelt werden kann, wurde in Unterkapitel 4.3.2 besprochen, die dort vorgestellte Methode wird hier mit einer Verzweigung optimiert.

Listing 19
Auszug aus 4-7_Seiten Durchlaufen-6.jsx

```
14 if (_seite.side == PageSideOptions.RIGHT_HAND) {
15   var _seitenzahlTF = _musterDruckbogen.pages[1].textFrames.
     itemByName("seitenzahl");
16 }
17 else {
18   var _seitenzahlTF = _musterDruckbogen.pages[0].textFrames.
     itemByName("seitenzahl");
19 }
```

`if (_seite.side == PageSideOptions.RIGHT_HAND) { //...`

14 In dieser Zeile befindet sich die Abfrage, ob die aktuelle Seite eine rechtsseitige ist. Die Eigenschaft `side` kann drei Werte annehmen:

- `PageSideOptions.LEFT_HAND`
- `PageSideOptions.RIGHT_HAND`
- `PageSideOptions.SINGLE_SIDED`

Verzweigung

14–19 Hier wird mit Hilfe einer *Verzweigung* die passende Musterseite auf dem Mustervorlagendruckbogen ermittelt.

Wenn in der for-Schleife gerade eine rechtsseitige Seite durchlaufen wird, wird auch der Textrahmen auf der rechtsseitigen Musterseite ausgewählt (`_musterDruckbogen.pages[1]` ...). Ansonsten wird die erste, also linksseitige Musterseite gewählt.

Im konkreten Beispiel gibt es drei Möglichkeiten: rechts, links oder nur eine einzelne Seite. Das heißt, im Falle einer rechten Seite wird der if-Block durchlaufen, im Falle einer linken oder einer einzelnen Seite wird der else-Block durchlaufen. Dies ist auch so gewollt, weil die Musterseite mit nur einer Seite ebenfalls auf `pages[0]` liegt.

Weitere Informationen zu Verzweigungen finden Sie in Unterkapitel 6.7.

4.8 Suchen und Ersetzen per Skript

Im folgenden Unterkapitel wird gezeigt, wie man per Skript die Suchen/Ersetzen-Funktion steuern kann.

Suchen/Ersetzen-Abfragen

Dazu verwende ich im ersten Skript Suchen/Ersetzen-Abfragen, die man im Suchen/Ersetzen-Dialog speichern kann. Die Abfragen stehen

programmweit für jedes Dokument zur Verfügung. Die genauen Optionen und Eigenschaften können bequem in der Registerkarte gesetzt und für die spätere Verwendung gespeichert werden. Im Skript wird dann eine bereits erstellte Abfrage geladen.

Um die Abfrage für das Skript zu erstellen, öffnen Sie den Suchen/Ersetzen-Dialog. Dort muss in der Registerkarte GREP die Suche nach dem Wort `Max` und die Änderung in `Moritz` erstellt werden. Diese Einstellungen werden als Suchen/Ersetzen-Abfragen mit dem Namen `MaxMoritz` gespeichert.

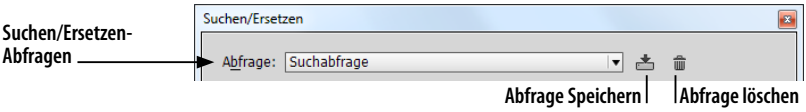


Abb. 35
Suchen/Ersetzen-Abfragen speichern

Diese Abfrage `MaxMoritz` wird dann im folgenden Skript geladen und vom Skript ausgeführt. Dieses Skript zeigt exemplarisch die Verwendung von Suchen/Ersetzen-Abfragen; produktiv wird es erst, wenn man mehrere Abfragen hintereinander ausführen lässt.

```
1  app.loadFindChangeQuery("MaxMoritz", SearchModes.GREP_SEARCH);
2  app.activeDocument.changeGrep();
```

Listing 20
4-8_SuchenUnd Ersetzen-1.jsx

1 Innerhalb des Skripts wird als Erstes die Abfrage `MaxMoritz` geladen. Die Methode `loadFindChangeQuery()` gehört zur Anwendung und wird direkt über `app` angesteuert. Die Methode übernimmt zwei Parameter. Der erste Parameter beinhaltet den Namen, der zweite Parameter den Typ der Abfrage. Für die vier verschiedenen Suchtypen muss einer der folgenden, sich selbst erklärenden Parameter verwendet werden:

Achten Sie auf die exakte Schreibweise des Namens.

- `SearchModes.TEXT_SEARCH`
- `SearchModes.GREP_SEARCH`
- `SearchModes.GLYPH_SEARCH`
- `SearchModes.OBJECT_SEARCH`

2 Die Suchen/Ersetzen-Abfrage wird mit der Methode `changeGrep()` im aktuell geöffneten Dokument ausgeführt. Die Methode muss ebenfalls mit dem Suchtyp korrespondieren.

- `changeText()` für die Text-Suche
- `changeGrep()` für die Suche mit GREP
- `changeGlyph()` für die Suche nach Glyphen
- `changeObject()` für die Suche nach Objekten

Die Einstellungen der Suche selbst setzen

Bei der Verwendung von Suchen/Ersetzen-Abfragen muss man sicherstellen, dass die Abfragen an dem Arbeitsplatz, an dem das Skript ausgeführt wird, verfügbar sind. Deswegen stelle ich in der Praxis oft die Eigenschaften für die gewünschte Abfrage direkt ein. Im Skript wird dann keine Abfrage geladen, sondern der Suchen/Ersetzen-Dialog di-

rekt per Skript gesteuert. Das folgende Skript zeigt, wie die Einstellungen für das Suchen und Ersetzen per GREP vorgenommen werden.

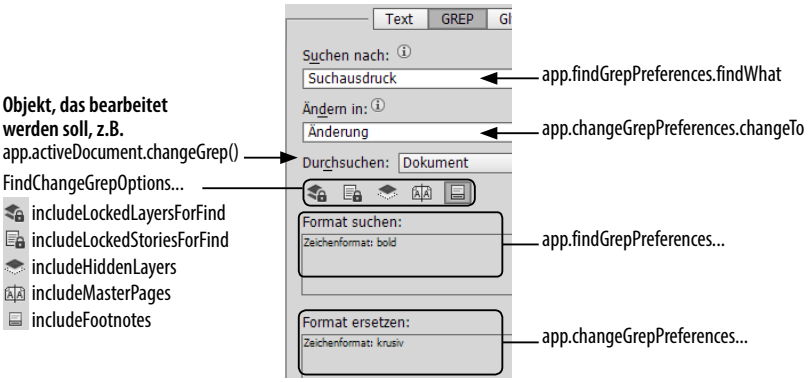
Listing 21
4-8_SuchenUnd Ersetzen-2.jsx

```
1 app.findGrepPreferences = NothingEnum.nothing;
2 app.findGrepPreferences.findWhat= "Max";
3 app.changeGrepPreferences = NothingEnum.nothing;
4 app.changeGrepPreferences.changeTo= "Moritz";
5 app.activeDocument.changeGrep();
```

```
app.findGrepPreferences = NothingEnum.nothing;
```

1 Die Eigenschaft `findGrepPreferences` gehört zur Anwendung. Hierüber können alle Such-Einstellungen gesetzt werden, die Sie auch im normalen Dialog eingeben können.

Abb. 36
Skripteinstellungen im Suchen/Ersetzen-Dialog



Als Erstes werden eventuell vorhandene Einstellungen im Bereich Suche mit dem Wert `NothingEnum.nothing` gelöscht. InDesign merkt sich die Einstellungen im Dialog, auch wenn dieser geschlossen ist. Dies ist im normalen Umgang mit der Funktion recht praktisch. Beim Skripting gibt es jedoch böse Überraschungen, wenn in den Such-Einstellungen noch Werte der vorherigen Suche gesetzt sind.

In Voreinstellungen (engl. preferences) können meist mehrere Werte gesetzt werden, deswegen stehen die Namen der Eigenschaften in der Pluralform. Es handelt sich hier aber ausnahmsweise nicht um Sammlungen. Der Aufruf ~~`app.findGrepPreferences[0]`~~ funktioniert nicht!

In ... Preferences sind keine Sammlungen enthalten.

```
app.findGrepPreferences.findWhat= "Max";
```

2 Die Eigenschaft `findWhat` in dieser Zeile entspricht dem Texteingabefeld SUCHEN NACH im Suchen/Ersetzen-Dialog. Es wird nach der Zeichenfolge `Max` gesucht.

```
app.changeGrepPreferences = NothingEnum.nothing;
```

3 Die Einstellungen für die Ersetzung müssen analog zu denen der Suche zurückgesetzt werden.

```
app.changeGrepPreferences.changeTo= "Moritz";
```

4 Die Eigenschaft `changeTo` entspricht dem Texteingabefeld ÄNDERN IN.

```
app.activeDocument.changeGrep();
```

5 Die Methode `changeGrep()` führt die Ersetzung im aktuellen Dokument aus, sie entspricht dem Button ALLE ÄNDERN aus dem Suchen/Ersetzen-Dialog. Die Methode kann von vielen Objekten aus gestartet werden, Sie können auch nur einen einzelnen Textrahmen oder Absatz durchsuchen.

Nach der Ausführung des Skripts stehen noch `Max` und `Moritz` in den entsprechenden Feldern des Suchen/Ersetzen-Dialogs. Um das zu verhindern, müssen die `findGrepPreferences` und `ChangeGrepPreferences` am Ende des Skripts erneut auf den Wert `NothingEnum.nothing` gesetzt werden.

InDesign »merkt« sich die Einstellungen.

Um ein Skript zu überprüfen, kann es aber durchaus hilfreich sein, die Werte nach Beendigung nicht zu löschen. So kann man nach Ablauf des Skripts bequem nachprüfen, ob die Anfrage aus dem Skript korrekt im Suchen/Ersetzen-Dialog angekommen ist.

Such-Optionen

Die Such-Optionen, also z.B. ob gesperrte Ebenen oder Musterseiten mit in die Suche aufgenommen werden sollen, sind Eigenschaften von `app.findChangeGrepOptions` und müssen bei Bedarf ebenfalls eingestellt werden. Dies wird in Unterkapitel 7.14 gezeigt.

Formate suchen und ersetzen

Im unteren Bereich des Suchen/Ersetzen-Dialogs können die Such-Einstellungen mit Formatangaben weiter verfeinert werden. Wenn nur Formatangaben und keine Suchbegriffe eingegeben werden, werden alle Textstellen, auf die die Formatangaben zutreffen, gefunden. Falls zusätzlich noch Formatangaben für die Änderung eingetragen werden, können die Formatierungen ersetzt werden.

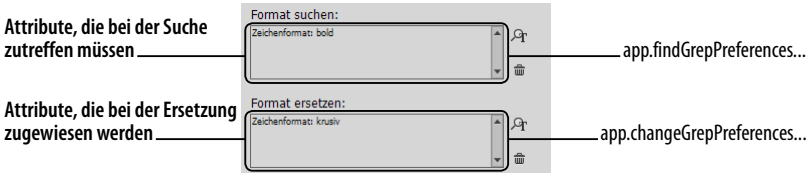


Abb. 37
Änderung der Formatangaben im Suchen/Ersetzen-Dialog

Diese Einstellungen können natürlich auch in Skripten verwendet werden. Das folgende Beispiel zeigt, wie man alle Textstellen, auf denen der Schriftschnitt `Italic` angewendet ist, mit dem Schriftschnitt `Bold` formatiert – also alle kursiven Texte in fett gesetzte Texte umwandelt. Dies setzt natürlich voraus, dass die Schriftschnitte der Schriftarten den entsprechenden Namen haben und vorhanden sind.

```
app.findGrepPreferences = NothingEnum.nothing;
app.findGrepPreferences.fontStyle = "Italic";
app.changeGrepPreferences = NothingEnum.nothing;
app.changeGrepPreferences.fontStyle = "Bold";
app.activeDocument.changeGrep();
```

Listing 22
4-8_SuchenUnd Ersetzen-3.jsx

```
app.findGrepPreferences.fontStyle = "Italic";
```

2 Mit der Eigenschaft `fontStyle` wird der Schriftschnitt für die Suchvorgaben festgelegt. Der Name muss als String übergeben werden.

```
app.changeGrepPreferences.fontStyle = "Bold";
```

4 Bei den Vorgaben für die Ersetzung wird ähnlich verfahren, wobei man auch noch zusätzlich Angaben für die Formatierung vergeben könnte. In der Suche bzw. bei der Ersetzung können alle Formateinstellungen, die auf Absatz- und Zeichenebene eingestellt werden können, verwendet werden. Die Wichtigsten finden Sie in der folgenden Tabelle. Die Eigenschaften müssen Sie wie in den oben gezeigten Skripten an `findGrepPreferences` bzw. `changeGrepPreferences` mit dem Punkt anhängen.

Tab. 11 Eigenschaften für Suchen/Ersetzen-Einstellungen

Eigenschaft	Beschreibung, findet/ ersetzt …
`findWhat = "Suche"`	Suchanfrage (nur `findGrepPreferences`)
`changeTo = "Ersetzung"`	Ersetzungsangabe (nur `changeGrepPreferences`)
`appliedCharacterStyle = "Formatname"`	das angewendete Zeichenformat
`appliedParagraphStyle "Formatname"`	das angewendete Absatzformat
`appliedFont = "Name der Schrift"`	die Schriftart
`fontStyle = "Schriftschnitt"`	den Schriftschnitt
`pointSize = 12`	die Schriftgröße
`position = Position.SUBSCRIPT`	tiefgestellten Text
`position = Position.SUPERSCRIPT`	hochgestellten Text
`capitalization = Capitalization.ALL_CAPS`	Versalien
`capitalization = Capitalization.SMALL_CAPS`	Kapitälchen
`fillColor = "Name der Farbe"`	Schriftfarbe
`justification = justification.LEFT_JUSTIFIED`	Blocksatz
`justification = Justification.RIGHT_ALIGN`	rechtsbündigen Text
`justification = Justification.LEFT_ALIGN`	linksbündigen Text
`justification = Justification.CENTER_ALIGN`	zentrierten Text

Leider kann man in der InDesign-Suche nicht negativ suchen, also nach Texten, die eine bestimmte Eigenschaft nicht aufweisen. In Unterkapitel 11.1.2 finden Sie eine Möglichkeit, nach Textstellen zu suchen, deren Werte außerhalb eines bestimmten Bereichs liegen.

4.9 Suchen, finden und verändern

Für komplexere Anforderungen reichen die Möglichkeiten der Suchen/Ersetzen-Funktion von InDesign nicht aus. Insbesondere wenn bestimmte Stellen in einem Dokument nicht einfach nur ersetzt, sondern nach bestimmten Regeln bearbeitet werden müssen, stößt die Funktion an ihre Grenzen. Die Einschränkungen können mit dem folgenden Skript aufgehoben werden. Es lässt sich immer einsetzen, wenn Sie bei der Ersetzung Berechnungen, Abfragen oder Veränderungen an der InDesign-Datei vornehmen müssen. Mit diesem Grundgerüst ließe sich also auch ein Skript für die Aktualisierung von Preisen anhand einer Liste oder die Umwandlung von Webadressen in echte Hyperlinks umsetzen.

Mit Suchergebnissen arbeiten

Die Idee für das Skript ist, die Suche wie gewohnt von InDesign ausführen zu lassen, die Ersetzung aber über eine eigene Funktion zu steuern. Für das Skript müssen die folgenden Schritte umgesetzt werden:

1. die Suchen/Ersetzen-Abfragen definieren
2. Anweisungen für die Ersetzung programmieren

In einem ersten Beispiel sollen alle Vorkommen des Wortes `Abbildung` innerhalb von Bildlegenden durch die Abkürzung `Abb.` ersetzt werden. Im normalen Text soll es weiterhin ausgeschrieben bleiben. Es wird vorausgesetzt, dass die Bildlegenden in Textrahmen stehen, die eindeutig durch das Objektformat `Legende` erkennbar sind. Als Beispieldokument können Sie die Datei *4-9_FindAndDo.idml* verwenden.

Das im Folgenden vorgestellte Skript setzt eine abgespeicherte GREP-Abfrage mit dem Namen `finde_Abbildung` voraus. Die Abfrage enthält lediglich die Suche nach dem Wort `Abbildung`.

Listing 23
4-9_FindAndDo-1.jsx

```
app.loadFindChangeQuery("finde_Abbildung", SearchModes.GREP_SEARCH);
var _ergebnisse = app.activeDocument.findGrep(true);
for (var i = 0; i < _ergebnisse.length; i++) {
  var _ergebnis = _ergebnisse[i];
  if (_ergebnis.parentTextFrames[0].appliedObjectStyle.name == "legende") {
    _ergebnis.contents = "Abb.";
  }
}
```

```
app.loadFindChangeQuery("finde_Abbildung", SearchModes.GREP_SEARCH);
```

1 Als Erstes wird die Suchabfrage `finde_Abbildung` mit der Methode `loadFindChangeQuery()` geladen. Achten Sie ab InDesign CC darauf, dass in der Suchabfrage die Suchrichtung VORWÄRTS aktiviert ist.

```
var _ergebnisse = app.activeDocument.findGrep(true);
```

2 Dann wird die Suche innerhalb des aktuellen Dokuments mit der Methode `findGrep()` ausgeführt und in der Variablen `_ergebnisse` abgelegt. Im Gegensatz zur Methode `changeGrep()` führt `findGrep()` nur die Suche aus. Die Ergebnisse der Suche werden als Array zurückgeliefert, jedes einzelne Ergebnis ist ein Textobjekt.

Mit `findGrep(true)` rückwärts suchen

Normalerweise wird ein Dokument vorwärts durchsucht. Alternativ kann man die Methode `findGrep()` auch anweisen, das Dokument rückwärts zu durchsuchen. Dazu muss man die Methode mit dem Parameter `true` aufrufen. Der Grund für die Rückwärtssuche ist, dass oftmals die Textreihenfolge im Dokument durch das Skript verändert wird. Wenn man vorne im Dokument die Textlänge verändert, werden die nachfolgenden Trefferstellen verschoben. InDesign würde nach der ersten Veränderung nur noch falsche Textstellen anspringen.

```
for (var i = 0; i < _ergebnisse.length; i++) {
   var _ergebnis = _ergebnisse[i];
   //...
}
```

3–8 Innerhalb der for-Schleife werden alle Ergebnisse durchlaufen. Zum Aufruf der einzelnen Ergebnisse aus dem Ergebnis-Array `_ergebnisse` wird die Zählvariable `i` verwendet. Das jeweilige Ergebnis wird in der Variablen `_ergebnis` abgelegt.

Den Textrahmen erreicht man über die Eigenschaft `parentTextFrames`.

```
if (_ergebnis.parentTextFrames[0].appliedObjectStyle.name == "legende"){
   //...
}
```

5–7 Der Vergleich der if-Abfrage prüft zunächst, ob der Textrahmen, in dem sich das Ergebnis befindet, mit dem Objektformat `legende` formatiert ist. Der dem Suchergebnis zugehörige Textrahmen wird nicht mit `parent`, sondern mit `parentTextFrames[0]` adressiert. Die Eigenschaft `parent` würde zum Textabschnitt, der sich aus den Inhalten der verketteten Rahmen zusammensetzt, führen. Das angewendete Objektformat kann über die Eigenschaft `appliedObjectStyle` abgerufen werden. Der Name des Objektformats wiederum befindet sich in der Eigenschaft `name`.

```
_ergebnis.contents = "Abb.";
```

6 Die Eigenschaft `contents` aller Suchergebnisse enthält den Wert `Abbildung`, denn danach wurde gesucht. In den einzelnen Ergebnissen wird der Inhalt mit dem gewünschten Wert überschrieben.

Die Abfrage des Objektformats ist nur eine von vielen Möglichkeiten. Denkbar wäre es auch, die Textrahmen danach zu prüfen, ob sie nur einen Absatz enthalten, innerhalb einer Gruppe mit einem Bild stehen oder andere bestimmte Eigenschaften aufweisen.

Formatabweichungen löschen

Mit diesem Skript als Grundgerüst kann die Suchfunktion fast beliebig ausgebaut werden. Das bekannte Skript *clearOverrides.jsx*, mit dem man alle Textstellen, deren Formatierung vom angewendeten Format abweicht, entfernen kann, basiert ebenfalls auf dieser Technik. Für dieses Kapitel habe ich eine etwas vereinfachte Version des Skripts geschrieben, für den produktiven Einsatz sollten Sie das Skript von meiner Homepage verwenden ↗130.

Im Skript werden einfach alle Textbereiche per GREP gesucht und eventuelle Abweichungen entfernt. Alternativ zu der Verwendung einer abgespeicherten Suchen/Ersetzen-Abfrage setze ich die Eigenschaften direkt im Skript.

Listing 24
Formatabweichungen löschen
4-9_clearOverrides.jsx

```
1  app.findGrepPreferences = NothingEnum.nothing;
2  app.findGrepPreferences.findWhat = "(?s).+";
3  var _ergebnisse = app.activeDocument.findGrep();
4  for (var i =0; i < _ergebnisse.length ; i++) {
5    _ergebnisse[i].clearOverrides();
6  }
7  app.findGrepPreferences = NothingEnum.nothing;
```

```
app.findGrepPreferences = NothingEnum.nothing;
```

1+7 Als Erstes werden eventuell vorhandene Such-Einstellungen in der Eigenschaft `findGrepPreferences` mit dem Wert `NothingEnum.nothing` gelöscht. Die im Skript eingestellten Such-Einstellungen werden ganz am Ende des Skripts in Zeile 7 ebenfalls wieder zurückgesetzt.

```
app.findGrepPreferences.findWhat = "(?s).+";
```

2 Mit dem GREP `(?s).+` findet man jedes beliebige Zeichen inklusive der Umbruchzeichen. Dies wird mit dem Modus `(?s)` erreicht, der das Sucherverhalten des Punkts entsprechend verändert. Details finden Sie im Unterkpaitel 1.2.6.

```
var _ergebnisse = app.activeDocument.findGrep();
```

3 Hier wird die Suche mit der Methode `findGrep()` auf dem aktiven Dokument ausgeführt. Die Methode liefert einen Array mit den Ergebnissen zurück, diese werden in der Variablen `_ergebnisse` gespeichert.

```
for (var i =0; i < _ergebnisse.length ; i++) {
   _ergebnisse[i].clearOverrides();
}
```

4–6 In einer for-Schleife werden alle Ergebnisse durchlaufen. Mit Hilfe der Methode `clearOverrides()` werden alle Abweichungen eines Ergebnisses in `_ergebnisse[i]` gelöscht. In der Suche wurden sämtliche Texte im Dokument gesucht, so dass am Ende der for-Schleife alle Abweichungen entfernt sind. Man könnte mit dem optionalen ersten Para-

meter noch steuern, ob nur Zeichen- oder Absatzformatabweichungen gelöscht werden sollen. Im ersten Fall übergibt man `OverrideType.CHARACTER_ONLY`, im zweiten `OverrideType.PARAGRAPH_ONLY`.

Seitenverweise verändern

Zum Abschluss noch ein etwas komplexeres Beispiel aus der Praxis: Alle Seitenverweise in einem Dokument wurden vom Auftraggeber »per Hand« gesetzt, nun verschob sich der Umbruch ab Seite 106 um vier Seiten. Mit der normalen Suchfunktion hätte das bedeutet, alle Seitenverweise von Hand durchzugehen und je nach Position zu verändern oder zu ignorieren. Bei 500 Verweisen wäre dies ziemlich zeitaufwändig geworden.

Das zugehörige Skript muss also zunächst alle Seitenverweise eindeutig auffinden und dann entscheiden, ob die Seitenzahl unverändert übernommen oder um 4 erhöht wird. Als Testdokument können Sie *4-9_Seitenverweise.idml* verwenden.

Listing 25
Seitenverweise verändern
4-9_FindAndDo-2.jsx

```
app.findGrepPreferences = NothingEnum.nothing;
if (app.findChangeGrepOptions.hasOwnProperty ("searchBackwards")) {
  app.findChangeGrepOptions.searchBackwards = false;
}
app.findGrepPreferences.findWhat = "\\(S\\.\\h\\d+\\)";
var _ergebnisse = app.activeDocument.findGrep(true);
app.findGrepPreferences.findWhat = "\\d+";
for (var i =0; i < _ergebnisse.length ; i++) {
  _ergebnis = _ergebnisse[i];
  var _ergebnis2 = _ergebnis.findGrep();
  var _zahl = parseFloat(_ergebnis2[0].contents);
  if (_zahl >= 106) {
    _zahl = _zahl + 4;
    _ergebnis.contents = "(S. " + _zahl + ")";
  }
}
app.findGrepPreferences = NothingEnum.nothing;
```

```
if (app.findChangeGrepOptions.hasOwnProperty ("searchBackwards")) {
   app.findChangeGrepOptions.searchBackwards = false;
}
```

Rückwärtssuche in CC deaktivieren

2–4 Ab InDesign CC kann Vorwärts und Rückwärts gesucht werden. Damit Skripte versionsunabhängig funktionieren, sollte man grundsätzlich die Richtung Vorwärts einstellen. Das gelingt in den Such-Optionen `findChangeGrepOptions` mit der Eigenschaft `searchBackwards`, die auf den Wert `false` gesetzt werden muss.

Auf die Eigenschaft `searchBackwards` kann jedoch nicht mit InDesign CS6 zugegriffen werden, so dass mit Hilfe einer if-Abfrage und der Methode `hasOwnProperty()` eine Versionsweiche eingebaut werden muss. Die Methode `hasOwnProperty()` prüft, ob das Objekt eine bestimmte Eigenschaft oder Methode besitzt. Weitere Details finden Sie

auf Seite 162. Wenn Sie ein Skript nur für InDesign CC entwickeln, können Sie die if-Abfrage natürlich weglassen und nur die Eigenschaft setzen.

```
app.findGrepPreferences.findWhat = "\\(S\\.\\h\\d+\\)";
```

Die Seitenverweise per GREP finden

5 Hier kommt jetzt das gesammelte Wissen über Reguläre Ausdrücke zum Einsatz. Um die Seitenverweise zu finden, wird der Ausdruck `\(S\.\h\d+\)` benötigt. Die runde Klammer muss mit einem Backslash maskiert werden, da nicht eine Rückwärtsreferenz benötigt wird, sondern nach den runden Klammern, die im Text vorkommen, gesucht werden soll. Das `S` wird buchstäblich gesucht. Der Punkt muss ebenfalls maskiert werden, weil der Punkt und nicht ein beliebiges Zeichen gesucht wird. Dazwischen steht ein Leerraum `\h`. Dann folgt die eigentliche Seitenzahl, die mit der Zeichenklasse `\d` und dem Wiederholungszeichen `+` gefunden werden soll. Am Schluss des Ausdrucks steht wieder die maskierte schließende runde Klammer.

Maskierung des Backslashs in JavaScript-Zeichenketten

Im Code des Skripts fallen die vielen doppelten Backslash-Zeichen auf. Sie werden benötigt, weil der Backslash im JavaScript-String eine besondere Bedeutung hat. Damit im Regulären Ausdruck ein Backslash ankommt, muss er bereits im JavaScript-String maskiert werden. Aus `\\` wird nach der Auflösung durch den JavaScript Interpreter `\`.

Da sich bei der doppelten Maskierung gerne Fehler einschleichen, empfehle ich für die Entwicklung von eigenen Skripten die Ausführung nach dem Setzen der Suchanweisung zu stoppen und dann im Suchen/Ersetzen-Dialog zu prüfen, ob die Suchanweisung richtig angekommen ist.

```
var _ergebnisse = app.activeDocument.findGrep(true);
```

Das Dokument rückwärts durchsuchen

6 Hier wird die Suche mit der Methode `findGrep()` auf dem aktiven Dokument ausgeführt. Mit dem Parameter `true` wird gesteuert, dass die Suche rückwärts ausgeführt wird. Das ist wichtig, weil die Textreihenfolge im Dokument durch das Skript verändert wird. Im vorherigen Skript war dies nicht der Fall und die Richtung der Suche deswegen unwichtig.

Lassen Sie sich nicht von der Such-Option Rückwärts, die weiter oben eingestellt wurde, verwirren. Diese ist nur in InDesign CC verfügbar und kann deswegen nicht für eine versionsunabhängige Rückwärtssuche eingesetzt werden. Vielmehr muss diese deaktiviert werden, so dass die Suchrichtung in InDesign CS6 und CC über die Methode gesteuert werden kann. Wenn Sie beide Male rückwärts aktivieren, würde im Endeffekt vorwärts gesucht. Die Methode liefert einen Array mit den Ergebnissen, die in der Variablen `_ergebnisse` gespeichert werden.

```
app.findGrepPreferences.findWhat = "\\d+";
```

7 Diese Zeile macht auf den ersten Blick keinen Sinn, da die Suche bereits durchgeführt wurde. In der for-Schleife muss jedoch nochmals jedes Suchergebnis nach der eigentlichen Seitenzahl durchsucht werden.

Das Suchergebnis enthält nicht nur die Seitenzahl, sondern die komplette Fundstelle, also z.B. `(S. 150)`. Die Such-Einstellungen stellt man aus Performancegründen jedoch nicht bei jedem Schleifendurchlauf, sondern einmalig vorher ein. Mit dem Regulären Ausdruck `\\d+` wird die eigentliche Zahl gefunden.

Aber warum muss überhaupt erneut gesucht werden? Man könnte doch auch eine Rückwärtsreferenz verwenden, also bei der Suche die Zahl gruppieren: `\(S\.\h(\d+)\)` und dann in der Ersetzung mit `$1` auf diese zugreifen (zur Verwendung von Rückwärtsreferenzen und Fundstellen siehe Unterkapitel 1.4).

Das Problem ist, dass bei der Suche mit der Methode `findGrep()` nur die vollständigen Treffer zurückgeliefert werden. Im Array `_ergebnisse` steht jeweils nur der vollständige Treffer, also `$0` – die einzelnen Fundstellen sind nicht enthalten.

Wenn die Methode `changeGrep()` für die Ersetzung verwendet würde, könnten Rückwärtsreferenzen verwendet werden. Da das Skript aber über die Möglichkeiten der normalen GREP-Ersetzung hinausgeht, muss hier die Ersetzung neu programmiert werden.

Eine Alternative ist die Verwendung von Look Around Assertions, die in Unterkapitel 10.6 beschrieben werden. Mit dem Suchausdruck `(?<=\(S\.\h)\d+(?=\))` könnte die zweite Suche in Zeile 10 entfallen. Das gekürzte Skript finden Sie in der Datei *4-9_FindAndDo-3.jsx*.

```
var _ergebnis2 = _ergebnis.findGrep();
```

10 Hier wird aus dem ersten Suchergebnis die eigentliche Seitenzahl mit der Such-Einstellung aus Zeile 7 ermittelt.

Einen String in eine Zahl verwandeln

```
var _zahl = parseFloat(_ergebnis2[0].contents);
```

11 Das Ergebnis der Suche in der Variablen `_ergebnis2` ist ein Objekt vom Typ Text. Der Inhalt `contents` enthält einen String. Dieser enthält zwar Ziffern, ist aber keine echte Zahl. Um eine Zahl zum Rechnen und Vergleichen zu erhalten, muss der Inhalt der Variablen mit `parseFloat()` in eine Zahl verwandelt werden. Das Ergebnis wird in der Variablen `_zahl` gespeichert.

```
if (_zahl >= 106) { //...
```

12–15 In der Abfrage wird geprüft, ob die in der Variablen `_zahl` gespeicherte Zahl größer oder gleich `106` ist. Wenn dies der Fall ist, wird der Anweisungsblock zwischen den geschweiften Klammern ausgeführt.

```
_zahl = _zahl + 4;
_ergebnis.contents = "(S. " + _zahl + ")";
```

13+14 Innerhalb des Anweisungsblocks wird der Wert der Variablen `_zahl` um `4` erhöht und an die Textstelle des Ergebnisses geschrieben.

Zahlen in Strings verwandeln

In den beiden Zeilen wird die unterschiedliche Funktion des +-Operators bei der Addition von Zahlen bzw. dem Zusammenfügen von Zeichenketten deutlich. Wenn es um Zahlen geht, ergeben `106 + 4 = 110`. Wenn es um Zeichenketten geht, ergibt sich aus `"(S. " + 110 + ")"` der Text `(S. 110)`. JavaScript verändert die Zahl 110 automatisch in einen String, wenn sie mit dem Plus-Zeichen zu einem anderen String hinzugefügt wird. Dies liegt daran, dass die Wertigkeit von Strings höher als die von Zahlen ist. Aus der Zeichenkette `"106"` plus der Zahl `4` ergibt sich der String `"1064"`.

Rechnen können Sie nur mit echten Zahlen, gegebenenfalls müssen Zeichenketten mit `parseFloat()` in Zahlen verwandelt werden. Details dazu finden Sie im Unterkapitel 6.5.

Auf welcher Seite bin ich?

Die Auswahl des Textrahmens mit dem Namen `seitenzahl` in Zeile 14 des letzten Skripts ist auf die linke Musterseite beschränkt. Falls auch mit dem Textrahmen auf der rechten Seite gearbeitet werden soll, benötigt man eine Möglichkeit, mit der entweder die rechte oder linke Musterseite ausgewählt werden kann. Wie die Position der Seite relativ zum Bund ermittelt werden kann, wurde in Unterkapitel 4.3.2 besprochen, die dort vorgestellte Methode wird hier mit einer Verzweigung optimiert.

Listing 19
Auszug aus 4-7_Seiten Durchlaufen-6.jsx

```
14 if (_seite.side == PageSideOptions.RIGHT_HAND) {
15   var _seitenzahlTF = _musterDruckbogen.pages[1].textFrames.
     itemByName("seitenzahl");
16 }
17 else {
18   var _seitenzahlTF = _musterDruckbogen.pages[0].textFrames.
     itemByName("seitenzahl");
19 }
```

`if (_seite.side == PageSideOptions.RIGHT_HAND) { //...`

14 In dieser Zeile befindet sich die Abfrage, ob die aktuelle Seite eine rechtsseitige ist. Die Eigenschaft `side` kann drei Werte annehmen:

- `PageSideOptions.LEFT_HAND`
- `PageSideOptions.RIGHT_HAND`
- `PageSideOptions.SINGLE_SIDED`

Verzweigung

14–19 Hier wird mit Hilfe einer *Verzweigung* die passende Musterseite auf dem Mustervorlagendruckbogen ermittelt.

Wenn in der for-Schleife gerade eine rechtsseitige Seite durchlaufen wird, wird auch der Textrahmen auf der rechtsseitigen Musterseite ausgewählt (`_musterDruckbogen.pages[1]` ...). Ansonsten wird die erste, also linksseitige Musterseite gewählt.

Im konkreten Beispiel gibt es drei Möglichkeiten: rechts, links oder nur eine einzelne Seite. Das heißt, im Falle einer rechten Seite wird der if-Block durchlaufen, im Falle einer linken oder einer einzelnen Seite wird der else-Block durchlaufen. Dies ist auch so gewollt, weil die Musterseite mit nur einer Seite ebenfalls auf `pages[0]` liegt.

Weitere Informationen zu Verzweigungen finden Sie in Unterkapitel 6.7.

4.8 Suchen und Ersetzen per Skript

Im folgenden Unterkapitel wird gezeigt, wie man per Skript die Suchen/Ersetzen-Funktion steuern kann.

Suchen/Ersetzen-Abfragen

Dazu verwende ich im ersten Skript Suchen/Ersetzen-Abfragen, die man im Suchen/Ersetzen-Dialog speichern kann. Die Abfragen stehen

programmweit für jedes Dokument zur Verfügung. Die genauen Optionen und Eigenschaften können bequem in der Registerkarte gesetzt und für die spätere Verwendung gespeichert werden. Im Skript wird dann eine bereits erstellte Abfrage geladen.

Um die Abfrage für das Skript zu erstellen, öffnen Sie den Suchen/Ersetzen-Dialog. Dort muss in der Registerkarte GREP die Suche nach dem Wort `Max` und die Änderung in `Moritz` erstellt werden. Diese Einstellungen werden als Suchen/Ersetzen-Abfragen mit dem Namen `MaxMoritz` gespeichert.

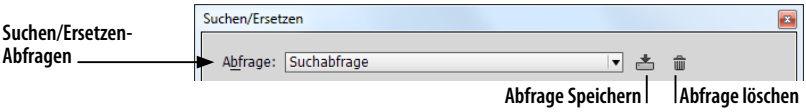


Abb. 35
Suchen/Ersetzen-Abfragen speichern

Diese Abfrage `MaxMoritz` wird dann im folgenden Skript geladen und vom Skript ausgeführt. Dieses Skript zeigt exemplarisch die Verwendung von Suchen/Ersetzen-Abfragen; produktiv wird es erst, wenn man mehrere Abfragen hintereinander ausführen lässt.

```
1  app.loadFindChangeQuery("MaxMoritz", SearchModes.GREP_SEARCH);
2  app.activeDocument.changeGrep();
```

Listing 20
4-8_SuchenUnd Ersetzen-1.jsx

1 Innerhalb des Skripts wird als Erstes die Abfrage `MaxMoritz` geladen. Die Methode `loadFindChangeQuery()` gehört zur Anwendung und wird direkt über `app` angesteuert. Die Methode übernimmt zwei Parameter. Der erste Parameter beinhaltet den Namen, der zweite Parameter den Typ der Abfrage. Für die vier verschiedenen Suchtypen muss einer der folgenden, sich selbst erklärenden Parameter verwendet werden:

Achten Sie auf die exakte Schreibweise des Namens.

- `SearchModes.TEXT_SEARCH`
- `SearchModes.GREP_SEARCH`
- `SearchModes.GLYPH_SEARCH`
- `SearchModes.OBJECT_SEARCH`

2 Die Suchen/Ersetzen-Abfrage wird mit der Methode `changeGrep()` im aktuell geöffneten Dokument ausgeführt. Die Methode muss ebenfalls mit dem Suchtyp korrespondieren.

- `changeText()` für die Text-Suche
- `changeGrep()` für die Suche mit GREP
- `changeGlyph()` für die Suche nach Glyphen
- `changeObject()` für die Suche nach Objekten

Die Einstellungen der Suche selbst setzen

Bei der Verwendung von Suchen/Ersetzen-Abfragen muss man sicherstellen, dass die Abfragen an dem Arbeitsplatz, an dem das Skript ausgeführt wird, verfügbar sind. Deswegen stelle ich in der Praxis oft die Eigenschaften für die gewünschte Abfrage direkt ein. Im Skript wird dann keine Abfrage geladen, sondern der Suchen/Ersetzen-Dialog di-

rekt per Skript gesteuert. Das folgende Skript zeigt, wie die Einstellungen für das Suchen und Ersetzen per GREP vorgenommen werden.

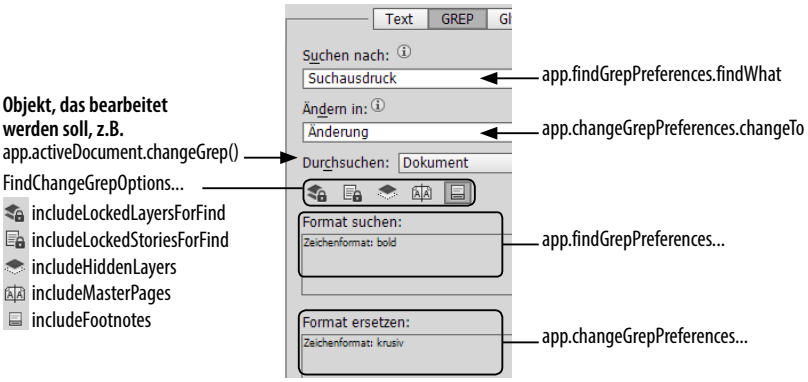
Listing 21
4-8_SuchenUnd Ersetzen-2.jsx

```
1 app.findGrepPreferences = NothingEnum.nothing;
2 app.findGrepPreferences.findWhat= "Max";
3 app.changeGrepPreferences = NothingEnum.nothing;
4 app.changeGrepPreferences.changeTo= "Moritz";
5 app.activeDocument.changeGrep();
```

```
app.findGrepPreferences = NothingEnum.nothing;
```

1 Die Eigenschaft `findGrepPreferences` gehört zur Anwendung. Hierüber können alle Such-Einstellungen gesetzt werden, die Sie auch im normalen Dialog eingeben können.

Abb. 36
Skripteinstellungen im Suchen/Ersetzen-Dialog



Als Erstes werden eventuell vorhandene Einstellungen im Bereich Suche mit dem Wert `NothingEnum.nothing` gelöscht. InDesign merkt sich die Einstellungen im Dialog, auch wenn dieser geschlossen ist. Dies ist im normalen Umgang mit der Funktion recht praktisch. Beim Skripting gibt es jedoch böse Überraschungen, wenn in den Such-Einstellungen noch Werte der vorherigen Suche gesetzt sind.

In Voreinstellungen (engl. preferences) können meist mehrere Werte gesetzt werden, deswegen stehen die Namen der Eigenschaften in der Pluralform. Es handelt sich hier aber ausnahmsweise nicht um Sammlungen. Der Aufruf ~~`app.findGrepPreferences[0]`~~ funktioniert nicht!

In ... Preferences sind keine Sammlungen enthalten.

```
app.findGrepPreferences.findWhat= "Max";
```

2 Die Eigenschaft `findWhat` in dieser Zeile entspricht dem Texteingabefeld SUCHEN NACH im Suchen/Ersetzen-Dialog. Es wird nach der Zeichenfolge `Max` gesucht.

```
app.changeGrepPreferences = NothingEnum.nothing;
```

3 Die Einstellungen für die Ersetzung müssen analog zu denen der Suche zurückgesetzt werden.

```
app.changeGrepPreferences.changeTo= "Moritz";
```

4 Die Eigenschaft `changeTo` entspricht dem Texteingabefeld ÄNDERN IN.

```
app.activeDocument.changeGrep();
```

5 Die Methode `changeGrep()` führt die Ersetzung im aktuellen Dokument aus, sie entspricht dem Button ALLE ÄNDERN aus dem Suchen/Ersetzen-Dialog. Die Methode kann von vielen Objekten aus gestartet werden, Sie können auch nur einen einzelnen Textrahmen oder Absatz durchsuchen.

Nach der Ausführung des Skripts stehen noch `Max` und `Moritz` in den entsprechenden Feldern des Suchen/Ersetzen-Dialogs. Um das zu verhindern, müssen die `findGrepPreferences` und `ChangeGrepPreferences` am Ende des Skripts erneut auf den Wert `NothingEnum.nothing` gesetzt werden.

InDesign »merkt« sich die Einstellungen.

Um ein Skript zu überprüfen, kann es aber durchaus hilfreich sein, die Werte nach Beendigung nicht zu löschen. So kann man nach Ablauf des Skripts bequem nachprüfen, ob die Anfrage aus dem Skript korrekt im Suchen/Ersetzen-Dialog angekommen ist.

Such-Optionen

Die Such-Optionen, also z.B. ob gesperrte Ebenen oder Musterseiten mit in die Suche aufgenommen werden sollen, sind Eigenschaften von `app.findChangeGrepOptions` und müssen bei Bedarf ebenfalls eingestellt werden. Dies wird in Unterkapitel 7.14 gezeigt.

Formate suchen und ersetzen

Im unteren Bereich des Suchen/Ersetzen-Dialogs können die Such-Einstellungen mit Formatangaben weiter verfeinert werden. Wenn nur Formatangaben und keine Suchbegriffe eingegeben werden, werden alle Textstellen, auf die die Formatangaben zutreffen, gefunden. Falls zusätzlich noch Formatangaben für die Änderung eingetragen werden, können die Formatierungen ersetzt werden.

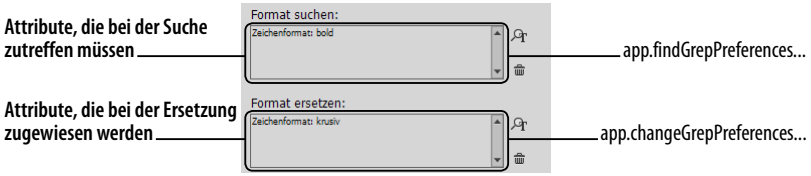


Abb. 37
Änderung der Formatangaben im Suchen/Ersetzen-Dialog

Diese Einstellungen können natürlich auch in Skripten verwendet werden. Das folgende Beispiel zeigt, wie man alle Textstellen, auf denen der Schriftschnitt `Italic` angewendet ist, mit dem Schriftschnitt `Bold` formatiert – also alle kursiven Texte in fett gesetzte Texte umwandelt. Dies setzt natürlich voraus, dass die Schriftschnitte der Schriftarten den entsprechenden Namen haben und vorhanden sind.

```
app.findGrepPreferences = NothingEnum.nothing;
app.findGrepPreferences.fontStyle = "Italic";
app.changeGrepPreferences = NothingEnum.nothing;
app.changeGrepPreferences.fontStyle = "Bold";
app.activeDocument.changeGrep();
```

Listing 22
4-8_SuchenUnd Ersetzen-3.jsx

```
app.findGrepPreferences.fontStyle = "Italic";
```

2 Mit der Eigenschaft `fontStyle` wird der Schriftschnitt für die Suchvorgaben festgelegt. Der Name muss als String übergeben werden.

```
app.changeGrepPreferences.fontStyle = "Bold";
```

4 Bei den Vorgaben für die Ersetzung wird ähnlich verfahren, wobei man auch noch zusätzlich Angaben für die Formatierung vergeben könnte. In der Suche bzw. bei der Ersetzung können alle Formateinstellungen, die auf Absatz- und Zeichenebene eingestellt werden können, verwendet werden. Die Wichtigsten finden Sie in der folgenden Tabelle. Die Eigenschaften müssen Sie wie in den oben gezeigten Skripten an `findGrepPreferences` bzw. `changeGrepPreferences` mit dem Punkt anhängen.

Tab. 11 Eigenschaften für Suchen/Ersetzen-Einstellungen

Eigenschaft	Beschreibung, findet/ ersetzt …
`findWhat = "Suche"`	Suchanfrage (nur `findGrepPreferences`)
`changeTo = "Ersetzung"`	Ersetzungsangabe (nur `changeGrepPreferences`)
`appliedCharacterStyle = "Formatname"`	das angewendete Zeichenformat
`appliedParagraphStyle "Formatname"`	das angewendete Absatzformat
`appliedFont = "Name der Schrift"`	die Schriftart
`fontStyle = "Schriftschnitt"`	den Schriftschnitt
`pointSize = 12`	die Schriftgröße
`position = Position.SUBSCRIPT`	tiefgestellten Text
`position = Position.SUPERSCRIPT`	hochgestellten Text
`capitalization = Capitalization.ALL_CAPS`	Versalien
`capitalization = Capitalization.SMALL_CAPS`	Kapitälchen
`fillColor = "Name der Farbe"`	Schriftfarbe
`justification = justification.LEFT_JUSTIFIED`	Blocksatz
`justification = Justification.RIGHT_ALIGN`	rechtsbündigen Text
`justification = Justification.LEFT_ALIGN`	linksbündigen Text
`justification = Justification.CENTER_ALIGN`	zentrierten Text

Leider kann man in der InDesign-Suche nicht negativ suchen, also nach Texten, die eine bestimmte Eigenschaft nicht aufweisen. In Unterkapitel 11.1.2 finden Sie eine Möglichkeit, nach Textstellen zu suchen, deren Werte außerhalb eines bestimmten Bereichs liegen.

4.9 Suchen, finden und verändern

Für komplexere Anforderungen reichen die Möglichkeiten der Suchen/Ersetzen-Funktion von InDesign nicht aus. Insbesondere wenn bestimmte Stellen in einem Dokument nicht einfach nur ersetzt, sondern nach bestimmten Regeln bearbeitet werden müssen, stößt die Funktion an ihre Grenzen. Die Einschränkungen können mit dem folgenden Skript aufgehoben werden. Es lässt sich immer einsetzen, wenn Sie bei der Ersetzung Berechnungen, Abfragen oder Veränderungen an der InDesign-Datei vornehmen müssen. Mit diesem Grundgerüst ließe sich also auch ein Skript für die Aktualisierung von Preisen anhand einer Liste oder die Umwandlung von Webadressen in echte Hyperlinks umsetzen.

Mit Suchergebnissen arbeiten

Die Idee für das Skript ist, die Suche wie gewohnt von InDesign ausführen zu lassen, die Ersetzung aber über eine eigene Funktion zu steuern. Für das Skript müssen die folgenden Schritte umgesetzt werden:

1. die Suchen/Ersetzen-Abfragen definieren
2. Anweisungen für die Ersetzung programmieren

In einem ersten Beispiel sollen alle Vorkommen des Wortes `Abbildung` innerhalb von Bildlegenden durch die Abkürzung `Abb.` ersetzt werden. Im normalen Text soll es weiterhin ausgeschrieben bleiben. Es wird vorausgesetzt, dass die Bildlegenden in Textrahmen stehen, die eindeutig durch das Objektformat `Legende` erkennbar sind. Als Beispieldokument können Sie die Datei *4-9_FindAndDo.idml* verwenden.

Das im Folgenden vorgestellte Skript setzt eine abgespeicherte GREP-Abfrage mit dem Namen `finde_Abbildung` voraus. Die Abfrage enthält lediglich die Suche nach dem Wort `Abbildung`.

Listing 23
4-9_FindAndDo-1.jsx

```
app.loadFindChangeQuery("finde_Abbildung", SearchModes.GREP_SEARCH);
var _ergebnisse = app.activeDocument.findGrep(true);
for (var i = 0; i < _ergebnisse.length; i++) {
  var _ergebnis = _ergebnisse[i];
  if (_ergebnis.parentTextFrames[0].appliedObjectStyle.name == "legende") {
    _ergebnis.contents = "Abb.";
  }
}
```

```
app.loadFindChangeQuery("finde_Abbildung", SearchModes.GREP_SEARCH);
```

1 Als Erstes wird die Suchabfrage `finde_Abbildung` mit der Methode `loadFindChangeQuery()` geladen. Achten Sie ab InDesign CC darauf, dass in der Suchabfrage die Suchrichtung VORWÄRTS aktiviert ist.

```
var _ergebnisse = app.activeDocument.findGrep(true);
```

2 Dann wird die Suche innerhalb des aktuellen Dokuments mit der Methode `findGrep()` ausgeführt und in der Variablen `_ergebnisse` abgelegt. Im Gegensatz zur Methode `changeGrep()` führt `findGrep()` nur die Suche aus. Die Ergebnisse der Suche werden als Array zurückgeliefert, jedes einzelne Ergebnis ist ein Textobjekt.

Mit `findGrep(true)` rückwärts suchen

Normalerweise wird ein Dokument vorwärts durchsucht. Alternativ kann man die Methode `findGrep()` auch anweisen, das Dokument rückwärts zu durchsuchen. Dazu muss man die Methode mit dem Parameter `true` aufrufen. Der Grund für die Rückwärtssuche ist, dass oftmals die Textreihenfolge im Dokument durch das Skript verändert wird. Wenn man vorne im Dokument die Textlänge verändert, werden die nachfolgenden Trefferstellen verschoben. InDesign würde nach der ersten Veränderung nur noch falsche Textstellen anspringen.

```
for (var i = 0; i < _ergebnisse.length; i++) {
   var _ergebnis = _ergebnisse[i];
   //...
}
```

3–8 Innerhalb der for-Schleife werden alle Ergebnisse durchlaufen. Zum Aufruf der einzelnen Ergebnisse aus dem Ergebnis-Array `_ergebnisse` wird die Zählvariable `i` verwendet. Das jeweilige Ergebnis wird in der Variablen `_ergebnis` abgelegt.

Den Textrahmen erreicht man über die Eigenschaft `parentTextFrames`.

```
if (_ergebnis.parentTextFrames[0].appliedObjectStyle.name == "legende"){
   //...
}
```

5–7 Der Vergleich der if-Abfrage prüft zunächst, ob der Textrahmen, in dem sich das Ergebnis befindet, mit dem Objektformat `legende` formatiert ist. Der dem Suchergebnis zugehörige Textrahmen wird nicht mit `parent`, sondern mit `parentTextFrames[0]` adressiert. Die Eigenschaft `parent` würde zum Textabschnitt, der sich aus den Inhalten der verketteten Rahmen zusammensetzt, führen. Das angewendete Objektformat kann über die Eigenschaft `appliedObjectStyle` abgerufen werden. Der Name des Objektformats wiederum befindet sich in der Eigenschaft `name`.

```
_ergebnis.contents = "Abb.";
```

6 Die Eigenschaft `contents` aller Suchergebnisse enthält den Wert `Abbildung`, denn danach wurde gesucht. In den einzelnen Ergebnissen wird der Inhalt mit dem gewünschten Wert überschrieben.

Die Abfrage des Objektformats ist nur eine von vielen Möglichkeiten. Denkbar wäre es auch, die Textrahmen danach zu prüfen, ob sie nur einen Absatz enthalten, innerhalb einer Gruppe mit einem Bild stehen oder andere bestimmte Eigenschaften aufweisen.

Formatabweichungen löschen

Mit diesem Skript als Grundgerüst kann die Suchfunktion fast beliebig ausgebaut werden. Das bekannte Skript *clearOverrides.jsx*, mit dem man alle Textstellen, deren Formatierung vom angewendeten Format abweicht, entfernen kann, basiert ebenfalls auf dieser Technik. Für dieses Kapitel habe ich eine etwas vereinfachte Version des Skripts geschrieben, für den produktiven Einsatz sollten Sie das Skript von meiner Homepage verwenden ↗130.

Im Skript werden einfach alle Textbereiche per GREP gesucht und eventuelle Abweichungen entfernt. Alternativ zu der Verwendung einer abgespeicherten Suchen/Ersetzen-Abfrage setze ich die Eigenschaften direkt im Skript.

Listing 24
Formatabweichungen löschen
4-9_clearOverrides.jsx

```
1  app.findGrepPreferences = NothingEnum.nothing;
2  app.findGrepPreferences.findWhat = "(?s).+";
3  var _ergebnisse = app.activeDocument.findGrep();
4  for (var i =0; i < _ergebnisse.length ; i++) {
5    _ergebnisse[i].clearOverrides();
6  }
7  app.findGrepPreferences = NothingEnum.nothing;
```

```
app.findGrepPreferences = NothingEnum.nothing;
```

1+7 Als Erstes werden eventuell vorhandene Such-Einstellungen in der Eigenschaft `findGrepPreferences` mit dem Wert `NothingEnum.nothing` gelöscht. Die im Skript eingestellten Such-Einstellungen werden ganz am Ende des Skripts in Zeile 7 ebenfalls wieder zurückgesetzt.

```
app.findGrepPreferences.findWhat = "(?s).+";
```

2 Mit dem GREP `(?s).+` findet man jedes beliebige Zeichen inklusive der Umbruchzeichen. Dies wird mit dem Modus `(?s)` erreicht, der das Sucherverhalten des Punkts entsprechend verändert. Details finden Sie im Unterkpaitel 1.2.6.

```
var _ergebnisse = app.activeDocument.findGrep();
```

3 Hier wird die Suche mit der Methode `findGrep()` auf dem aktiven Dokument ausgeführt. Die Methode liefert einen Array mit den Ergebnissen zurück, diese werden in der Variablen `_ergebnisse` gespeichert.

```
for (var i =0; i < _ergebnisse.length ; i++) {
   _ergebnisse[i].clearOverrides();
}
```

4–6 In einer for-Schleife werden alle Ergebnisse durchlaufen. Mit Hilfe der Methode `clearOverrides()` werden alle Abweichungen eines Ergebnisses in `_ergebnisse[i]` gelöscht. In der Suche wurden sämtliche Texte im Dokument gesucht, so dass am Ende der for-Schleife alle Abweichungen entfernt sind. Man könnte mit dem optionalen ersten Para-

meter noch steuern, ob nur Zeichen- oder Absatzformatabweichungen gelöscht werden sollen. Im ersten Fall übergibt man `OverrideType.CHARACTER_ONLY`, im zweiten `OverrideType.PARAGRAPH_ONLY`.

Seitenverweise verändern

Zum Abschluss noch ein etwas komplexeres Beispiel aus der Praxis: Alle Seitenverweise in einem Dokument wurden vom Auftraggeber »per Hand« gesetzt, nun verschob sich der Umbruch ab Seite 106 um vier Seiten. Mit der normalen Suchfunktion hätte das bedeutet, alle Seitenverweise von Hand durchzugehen und je nach Position zu verändern oder zu ignorieren. Bei 500 Verweisen wäre dies ziemlich zeitaufwändig geworden.

Das zugehörige Skript muss also zunächst alle Seitenverweise eindeutig auffinden und dann entscheiden, ob die Seitenzahl unverändert übernommen oder um 4 erhöht wird. Als Testdokument können Sie *4-9_Seitenverweise.idml* verwenden.

Listing 25
Seitenverweise verändern
4-9_FindAndDo-2.jsx

```
app.findGrepPreferences = NothingEnum.nothing;
if (app.findChangeGrepOptions.hasOwnProperty ("searchBackwards")) {
  app.findChangeGrepOptions.searchBackwards = false;
}
app.findGrepPreferences.findWhat = "\\(S\\.\\h\\d+\\)";
var _ergebnisse = app.activeDocument.findGrep(true);
app.findGrepPreferences.findWhat = "\\d+";
for (var i =0; i < _ergebnisse.length ; i++) {
  _ergebnis = _ergebnisse[i];
  var _ergebnis2 = _ergebnis.findGrep();
  var _zahl = parseFloat(_ergebnis2[0].contents);
  if (_zahl >= 106) {
    _zahl = _zahl + 4;
    _ergebnis.contents = "(S. " + _zahl + ")";
  }
}
app.findGrepPreferences = NothingEnum.nothing;
```

```
if (app.findChangeGrepOptions.hasOwnProperty ("searchBackwards")) {
   app.findChangeGrepOptions.searchBackwards = false;
}
```

Rückwärtssuche in CC deaktivieren

2–4 Ab InDesign CC kann Vorwärts und Rückwärts gesucht werden. Damit Skripte versionsunabhängig funktionieren, sollte man grundsätzlich die Richtung Vorwärts einstellen. Das gelingt in den Such-Optionen `findChangeGrepOptions` mit der Eigenschaft `searchBackwards`, die auf den Wert `false` gesetzt werden muss.

Auf die Eigenschaft `searchBackwards` kann jedoch nicht mit InDesign CS6 zugegriffen werden, so dass mit Hilfe einer if-Abfrage und der Methode `hasOwnProperty()` eine Versionsweiche eingebaut werden muss. Die Methode `hasOwnProperty()` prüft, ob das Objekt eine bestimmte Eigenschaft oder Methode besitzt. Weitere Details finden Sie

auf Seite 162. Wenn Sie ein Skript nur für InDesign CC entwickeln, können Sie die if-Abfrage natürlich weglassen und nur die Eigenschaft setzen.

```
app.findGrepPreferences.findWhat = "\\(S\\.\\h\\d+\\)";
```

Die Seitenverweise per GREP finden

5 Hier kommt jetzt das gesammelte Wissen über Reguläre Ausdrücke zum Einsatz. Um die Seitenverweise zu finden, wird der Ausdruck `\(S\.\h\d+\)` benötigt. Die runde Klammer muss mit einem Backslash maskiert werden, da nicht eine Rückwärtsreferenz benötigt wird, sondern nach den runden Klammern, die im Text vorkommen, gesucht werden soll. Das `S` wird buchstäblich gesucht. Der Punkt muss ebenfalls maskiert werden, weil der Punkt und nicht ein beliebiges Zeichen gesucht wird. Dazwischen steht ein Leerraum `\h`. Dann folgt die eigentliche Seitenzahl, die mit der Zeichenklasse `\d` und dem Wiederholungszeichen `+` gefunden werden soll. Am Schluss des Ausdrucks steht wieder die maskierte schließende runde Klammer.

Maskierung des Backslashs in JavaScript-Zeichenketten

Im Code des Skripts fallen die vielen doppelten Backslash-Zeichen auf. Sie werden benötigt, weil der Backslash im JavaScript-String eine besondere Bedeutung hat. Damit im Regulären Ausdruck ein Backslash ankommt, muss er bereits im JavaScript-String maskiert werden. Aus `\\` wird nach der Auflösung durch den JavaScript Interpreter `\`.

Da sich bei der doppelten Maskierung gerne Fehler einschleichen, empfehle ich für die Entwicklung von eigenen Skripten die Ausführung nach dem Setzen der Suchanweisung zu stoppen und dann im Suchen/Ersetzen-Dialog zu prüfen, ob die Suchanweisung richtig angekommen ist.

```
var _ergebnisse = app.activeDocument.findGrep(true);
```

Das Dokument rückwärts durchsuchen

6 Hier wird die Suche mit der Methode `findGrep()` auf dem aktiven Dokument ausgeführt. Mit dem Parameter `true` wird gesteuert, dass die Suche rückwärts ausgeführt wird. Das ist wichtig, weil die Textreihenfolge im Dokument durch das Skript verändert wird. Im vorherigen Skript war dies nicht der Fall und die Richtung der Suche deswegen unwichtig.

Lassen Sie sich nicht von der Such-Option Rückwärts, die weiter oben eingestellt wurde, verwirren. Diese ist nur in InDesign CC verfügbar und kann deswegen nicht für eine versionsunabhängige Rückwärtssuche eingesetzt werden. Vielmehr muss diese deaktiviert werden, so dass die Suchrichtung in InDesign CS6 und CC über die Methode gesteuert werden kann. Wenn Sie beide Male rückwärts aktivieren, würde im Endeffekt vorwärts gesucht. Die Methode liefert einen Array mit den Ergebnissen, die in der Variablen `_ergebnisse` gespeichert werden.

```
app.findGrepPreferences.findWhat = "\\d+";
```

7 Diese Zeile macht auf den ersten Blick keinen Sinn, da die Suche bereits durchgeführt wurde. In der for-Schleife muss jedoch nochmals jedes Suchergebnis nach der eigentlichen Seitenzahl durchsucht werden.

Das Suchergebnis enthält nicht nur die Seitenzahl, sondern die komplette Fundstelle, also z.B. `(S. 150)`. Die Such-Einstellungen stellt man aus Performancegründen jedoch nicht bei jedem Schleifendurchlauf, sondern einmalig vorher ein. Mit dem Regulären Ausdruck `\\d+` wird die eigentliche Zahl gefunden.

Aber warum muss überhaupt erneut gesucht werden? Man könnte doch auch eine Rückwärtsreferenz verwenden, also bei der Suche die Zahl gruppieren: `\(S\.\h(\d+)\)` und dann in der Ersetzung mit `$1` auf diese zugreifen (zur Verwendung von Rückwärtsreferenzen und Fundstellen siehe Unterkapitel 1.4).

Das Problem ist, dass bei der Suche mit der Methode `findGrep()` nur die vollständigen Treffer zurückgeliefert werden. Im Array `_ergebnisse` steht jeweils nur der vollständige Treffer, also `$0` – die einzelnen Fundstellen sind nicht enthalten.

Wenn die Methode `changeGrep()` für die Ersetzung verwendet würde, könnten Rückwärtsreferenzen verwendet werden. Da das Skript aber über die Möglichkeiten der normalen GREP-Ersetzung hinausgeht, muss hier die Ersetzung neu programmiert werden.

Eine Alternative ist die Verwendung von Look Around Assertions, die in Unterkapitel 10.6 beschrieben werden. Mit dem Suchausdruck `(?<=\(S\.\h)\d+(?=\))` könnte die zweite Suche in Zeile 10 entfallen. Das gekürzte Skript finden Sie in der Datei *4-9_FindAndDo-3.jsx*.

```
var _ergebnis2 = _ergebnis.findGrep();
```

10 Hier wird aus dem ersten Suchergebnis die eigentliche Seitenzahl mit der Such-Einstellung aus Zeile 7 ermittelt.

Einen String in eine Zahl verwandeln

```
var _zahl = parseFloat(_ergebnis2[0].contents);
```

11 Das Ergebnis der Suche in der Variablen `_ergebnis2` ist ein Objekt vom Typ Text. Der Inhalt `contents` enthält einen String. Dieser enthält zwar Ziffern, ist aber keine echte Zahl. Um eine Zahl zum Rechnen und Vergleichen zu erhalten, muss der Inhalt der Variablen mit `parseFloat()` in eine Zahl verwandelt werden. Das Ergebnis wird in der Variablen `_zahl` gespeichert.

```
if (_zahl >= 106) { //...
```

12–15 In der Abfrage wird geprüft, ob die in der Variablen `_zahl` gespeicherte Zahl größer oder gleich `106` ist. Wenn dies der Fall ist, wird der Anweisungsblock zwischen den geschweiften Klammern ausgeführt.

```
_zahl = _zahl + 4;
_ergebnis.contents = "(S. " + _zahl + ")";
```

13+14 Innerhalb des Anweisungsblocks wird der Wert der Variablen `_zahl` um `4` erhöht und an die Textstelle des Ergebnisses geschrieben.

Zahlen in Strings verwandeln

In den beiden Zeilen wird die unterschiedliche Funktion des +-Operators bei der Addition von Zahlen bzw. dem Zusammenfügen von Zeichenketten deutlich. Wenn es um Zahlen geht, ergeben `106 + 4 = 110`. Wenn es um Zeichenketten geht, ergibt sich aus `"(S. " + 110 + ")"` der Text `(S. 110)`. JavaScript verändert die Zahl 110 automatisch in einen String, wenn sie mit dem Plus-Zeichen zu einem anderen String hinzugefügt wird. Dies liegt daran, dass die Wertigkeit von Strings höher als die von Zahlen ist. Aus der Zeichenkette `"106"` plus der Zahl `4` ergibt sich der String `"1064"`.

Rechnen können Sie nur mit echten Zahlen, gegebenenfalls müssen Zeichenketten mit `parseFloat()` in Zahlen verwandelt werden. Details dazu finden Sie im Unterkapitel 6.5.

verändert wurde, und speichert die Dateien im Backup-Verzeichnis in der Form *Kopierdatum_Änderungsdatum__Name.indd*.

`_dok.save();`
14 Zuletzt muss noch das aktuelle Dokument gespeichert werden. Dies geschieht mit der Methode `save()`.

Im nächsten Kapitel zeige ich, wie man einen Menü-Eintrag für das Skript erstellt und es beim Starten von InDesign automatisch aktivieren kann.

11.13 Eigene Einträge im Menü erstellen

In InDesign kann man eigene Menü-Einträge für Skripte erstellen. So können Sie Ihre Skripte in die InDesign-Menü-Struktur einhängen und müssen sie nicht mehr über das Bedienfeld Skripten aufrufen. Wenn man dann noch die Möglichkeit nutzt, Skripte direkt beim Start von InDesign zu laden, wird die Bedienung der Skripte deutlich angenehmer.

Gerade wenn Sie Skripte für andere Benutzer entwickeln, sind diese Techniken wichtig. Es hat sich gezeigt, dass Akzeptanz und Benutzung von Skripten in InDesign deutlich zunehmen, wenn sie in der normalen Menü-Struktur verankert sind.

11.13.1 Menü-Einträge

Als Beispiel zeige ich, wie man das Skript zum Erstellen eines Backups aus dem vorherigen Unterkapitel in das Datei-Menü hinter den Eintrag Speichern einhängen kann.

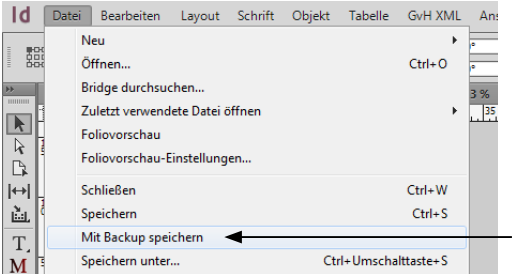


Abb. 101
Datei-Menü

Listing 124
Menü-Eintrag erstellen
11-13_MenuEintrag-1.jsx

```
#targetengine "saveWithBackupSession"
var _action = app.scriptMenuActions.add("Mit Backup speichern");
_action.eventListeners.add("onInvoke", saveWithBackup);
var _hm = app.menus.itemByName("$ID/Main");
var _dm = _hm.submenus.itemByName("$ID/FileDestinationPanel");
if (_dm.menuItems.itemByName("Mit Backup speichern") == null) {
  _dm.menuItems.add(_action, LocationOptions.AFTER, _dm.menuItems.
  itemByName("$ID/Save"));
}
```

```
#targetengine "saveWithBackupSession"
```

Session anlegen

1 Mit der Festlegung einer eigenen Session (→ Unterkapitel 7.17) stellt man sicher, dass das Skript bis zum Beenden von InDesign erreichbar ist. Variablen und eigene Funktionen sind nur innerhalb einer Session gültig und können deswegen nicht von anderen Skripten beeinflusst werden. Sobald das Skript einmal ausgeführt wurde, erscheint die Session `saveWithBackupSession` in der zweiten Dropdown-Liste des Debug-Bedienfelds vom ESTK. Bei Bedarf könnte man in einem anderen Skript, das in der gleichen Session ausgeführt wird, auf die Funktion `saveWithBackup()` zugreifen.

```
var _action = app.scriptMenuActions.add("Mit Backup speichern");
_action.eventListeners.add("onInvoke", saveWithBackup);
```

Menü-Befehl erstellen

2+3 Zunächst wird ein Objekt vom Typ `ScriptMenuAction`, das zum Objekt `Application` gehört, erstellt. Der Parameter der Methode `add()` übernimmt den Namen des Menü-Eintrags.

Dann muss dem Objekt noch ein `EventListener` hinzugefügt werden. Dieser soll immer dann, wenn der Menü-Eintrag ausgewählt wird, reagieren. Bei der Erstellung werden zwei Parameter übergeben: Der erste beschreibt das Event, der zweite, welche Funktion beim Eintreten des Events ausgeführt werden soll (zu Events siehe Unterkapitel 7.18). Das Event `onInvoke` tritt ein, wenn der Menü-Eintrag ausgewählt wird. Die Funktion `saveWithBackup()` ist weiter unten im Skript enthalten und entspricht weitgehend der Funktion aus dem vorherigen Unterkapitel.

Jetzt ist ein fertiger Menü-Eintrag erstellt, der allerdings noch nicht in ein Menü eingehangen wurde.

```
var _hm = app.menus.itemByName("$ID/Main");
```

4 In der Variablen `_hm` wird eine Referenz auf das Hauptmenü gespeichert. Das Hauptmenü erreicht man über die Sammlung `menus`. Die Menüs werden über die sprachunabhängigen Namen (engl. *locale independent string*) angesprochen. Wenn das Skript nur unter der deutschen Version von InDesign laufen würde, könnte man auch die Namen aus der Benutzeroberfläche übernehmen.

Menü-Einträge ansprechen

Generell sollten Menü-Einträge immer mit den sprachunabhängigen Namen und Menü-Befehle über ihre `ID` angesprochen werden. Da es keine Liste für diese Bezeichnungen gibt, kann man mit dem Skript *11-13_AlleMenus.jsx* eine Textdatei mit den Namen und IDs aller Menüs und Befehle erstellen (die Laufzeit kann mehrere Minuten betragen). Leider werden einige wenige Namen nicht korrekt aufgelöst, andere Menü-Einträge haben mehrere Entsprechungen. Außerdem haben sich einige Bezeichnungen in den verschiedenen Versionen geändert. Hier muss man im Zweifel etwas experimentieren.

```
var _dm = _hm.submenus.itemByName("$ID/FileDestinationPanel");
```

5 Mit Hilfe der Sammlung `submenus`, die die Untermenüs DATEI, BEARBEITEN etc. enthält, wird eine Referenz auf das Datei-Menü, das auf die sprachunabhängige Bezeichnung `$ID/FileDestinationPanel` hört, in der Variablen `_dm` gespeichert werden.

Menü-Eintrag erstellen

```
if (_dm.menuItems.itemByName("Mit Backup speichern") == null) {
   _dm.menuItems.add(_action, LocationOptions.AFTER, _dm.menuItems.
   itemByName("$ID/Save"));
}
```

6–8 Zunächst wird geprüft, ob der Eintrag MIT BACKUP SPEICHERN bereits vorhanden ist. Wenn dies nicht der Fall ist, wird der Menü-Eintrag erstellt. Dazu wird dem Datei-Menü ein neues Objekt vom Typ `MenuItem` hinzugefügt. Als erster Parameter wird die `ScriptMenuAction`, die vorher in der Variablen `_action` gespeichert wurde, übergeben. Der zweite Parameter legt die Position des neuen Elements in Bezug zum dritten Parameter – einem Menü-Eintrag – fest. Der neue Menü-Eintrag erscheint unter dem Eintrag SPEICHERN.

Menü-Eintrag kontextabhängig einblenden

Das Skript aus dem vorherigen Unterkapitel prüft zu Beginn, ob es sinnvoll ausführbar ist. Diese Prüfung kann man auch in eine eigene Funktion auslagern, die je nach Situation den Menü-Eintrag ein- bzw. ausblendet.

Listing 125 Menü-Eintrag kontextabhängig *11-13_Menu Eintrag-2.jsx*

```
2  var _action = app.scriptMenuActions.add("Mit Backup speichern");
3  _action.eventListeners.add("onInvoke", saveWithBackup);
4  _action.eventListeners.add("beforeDisplay", canRun);
   //...
22 function canRun(_event) {
23   var _action = _event.parent;
24   if (app.documents.length > 0 && app.activeDocument.saved == true
     && app.activeDocument.modified == true) {
25     _action.enabled = true;
26   } else {
27     _action.enabled = false;
28   }
29 }
```

4 Für die Prüfung, ob der Eintrag angezeigt werden soll, kann man einen Eventlistener vom Typ `beforeDisplay` verwenden. Bevor der Eintrag angezeigt wird, wird dann immer die Funktion `canRun()` ausgeführt.

```
function canRun(_event) { //...
```

22–29 Der Funktion `canRun()` wird von InDesign automatisch das Event, von dem sie ausgelöst wurde, als Parameter übergeben. In Zeile 23 wird dann die eigentliche `ScriptMenuAction` ermittelt – sie ist das Elternelement des Events.

```
if (app.documents.length > 0 && app.activeDocument.saved == true
   && app.activeDocument.modified == true) {
```

24 Hier wird geprüft, ob die Funktion sinnvoll ausgeführt werden kann – die Prüfung ist im ursprünglichen Skript *11-12_SaveWith Backup-1.jsx* in der ersten Zeile durchgeführt worden. Wenn die Prüfung erfolgreich ist, wird die `ScriptMenuAction` mit der Eigenschaft `enabled` aktiviert, ansonsten deaktiviert.

Menü-Eintrag entfernen

Der Menü-Eintrag bleibt nur bis zu einem Neustart von InDesign erhalten. Wenn Sie den Eintrag vorher entfernen wollen, benötigen Sie das folgende Skript:

```
4  var _eintrag = _dm.menuItems.itemByName("Mit Backup speichern");
5  if (_eintrag != null) {
6    _eintrag.remove();
7  }
```

Listing 126
Menü-Eintrag entfernen
11-13_MenuEintrag Entfernen.jsx

Das Skript muss ebenfalls in der Session `saveWithBackupSession` laufen. In Zeile 2+3 wird wie im vorherigen Skript das Datei-Menü adressiert. Danach wird der Menü-Eintrag in `_eintrag` gespeichert und mit einer if-Abfrage geprüft, ob der Eintrag im Datei-Menü existiert. Wenn `_eintrag` dem Wert `null` entspricht, existiert kein Eintrag. Mit der Methode `remove()` kann der Eintrag gegebenenfalls entfernt werden.

Untermenüs bleiben erhalten.

Wenn Sie eigene Untermenüs (`Submenu`) erstellen, bleiben diese auch nach einem Neustart erhalten. Die folgende Zeile erstellt ein Untermenü Skripte am Ende der Menü-Leiste.

```
app.menus.itemByName("$ID/Main").submenus.add("Skripte");
```

Zum Entfernen des Menüs können Sie die folgende Zeile verwenden:

```
app.menus.itemByName("$ID/Main").submenus.itemByName("Skripte").remove();
```

11.13.2 Skripte beim Start von InDesign laden

Für eine gute Integration des Skripts in InDesign sollte das Skript automatisch beim Start geladen und der Menü-Eintrag erstellt werden.

Dazu muss das Skript in einen Ordner mit dem Namen *startup scripts* gelegt werden. Die Startskripte von InDesign, wie z. B. *URLs in Hyperlinks konvertieren*, liegen neben dem Ordner *Scripts Panel*. Ich empfehle, eigene Skripte auf der gleichen Ebene zu speichern. Für das Skript würde ich deswegen erst einen Ordner *SaveWithBackup* und in diesem dann den Unterordner *startup scripts* anlegen. Nachdem das Skript hierhin kopiert wurde, brauchen Sie nur noch InDesign neu zu starten.

Wenn man das Skript regelmäßig verwenden möchte, kann man es auf einen Tastaturbefehl legen (Bearbeiten → Tastaturbefehle … → Produktbereich Skripte).

den Alternativtext im Dialog Objektexportoptionen eintragen (Objekt → Objektexportoptionen). Der Alternativtext wird auch für den Export einer barrierefreien PDF-Datei benötigt.

Die gerade genannten Anforderungen für Bilder können mit dem Skript in Unterkapitel 13.7 geprüft werden.

Dateinamen

Leer- und Sonderzeichen in den Dateinamen von verknüpften Dateien können zu Problemen bei älteren Lesegeräten führen. Um dem vorzubeugen, kann man leicht mit dem Skript *relinkImages.jsx* aus Unterkapitel 13.4.2 die Dateinamen bereinigen.

13.2 Problematische Zeichen prüfen

Durch den Neuumbruch im E-Book können manuelle Trennungen und Umbrüche, die für die Druckausgabe eingefügt wurden, sichtbar werden. Wenn Sie viele E-Books lesen, ist Ihnen dieses Problem vermutlich schon aufgefallen. Die folgenden Arbeitstechniken führen zu Problemen beim E-Book-Export und sollten bereits beim Dokumentaufbau vermieden werden:

- Trennungen, die durch Bindestriche ausgeführt werden, bleiben im Export erhalten. Verwenden Sie ausschließlich *bedingte Trennstriche*.
- *Harte Zeilenumbrüche* sollten nur dann eingesetzt werden, wenn diese auch im Export enthalten sein sollen.
- *Tabulatoren* und das Zeichen *Einzug hier* können nicht dargestellt werden. Das gilt insbesondere für den Aufbau von *Listen*.
- *Leerzeilen* oder *Leerzeichen* sollten nicht zur Formatierung eingesetzt werden.

Eine automatische Korrektur der oben genannten Punkte ist nicht möglich, da insbesondere Trennungen durch Bindestriche nicht eindeutig von Koppelwörtern bzw. Zusammenschreibungen unterschieden werden können. Aber auch die anderen Zeichen bedürfen normalerweise einer manuellen Korrektur. Per Skript lassen sich aber die problematischen Stellen im Dokument auffinden und dann bei Bedarf korrigieren.

Interaktive Korrektur

Das Skript *findCriticalText.jsx* erledigt diesen Job. Es springt zu den entsprechenden Textstellen und bietet die Möglichkeit, für jeden Einzelfall zu entscheiden. Der Fokus liegt auf der interaktiven Bearbeitung der gefundenen Fehlerstellen, das Skript beinhaltet zusätzlich die Möglichkeit, Suchen/Ersetzen-Funktionen generell anzuwenden. Zur Sicherheit wird das Dokument vor der Korrekturphase gespeichert. Zum Testen können Sie das Dokument *findCriticalText.idml* verwenden. Das Skript ist auch für die Konvertierung in andere digitale Ausgabeformate hilfreich.

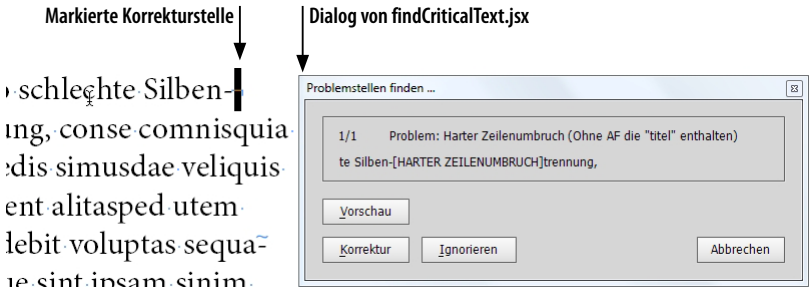


Abb. 108
Korrekturfenster von *findCriticalText.jsx*

Korrekturen durchführen

Wenn Sie den Button KORREKTUR wählen, wird die Korrektur ausgeführt, mit IGNORIEREN wird die Fehlerstelle nicht korrigiert. In beiden Fällen springt das Skript danach automatisch zur nächsten Fehlerstelle. Wenn Sie VORSCHAU anklicken, wird die Korrektur im Dokument ausgeführt, die Auswahl bleibt bei der aktuellen Fehlerstelle. Wenn Sie danach IGNORIEREN wählen, wird die Korrektur rückgängig gemacht, wenn Sie KORREKTUR anklicken, bleibt die Korrektur bestehen.

Veränderungen vornehmen

Der Dialog ist nicht modal, so dass Sie auch direkt im Dokument eingreifen können. Sie dürfen allerdings nicht in den Textbereich vor der Fehlerstelle eingreifen, da sich dann ggf. der Index der Fehlerstelle verschiebt und die Korrekturfunktion bei Folgefehlern nicht mehr zuverlässig arbeitet. Format- oder Textänderungen, die keinen Einfluss auf den Index haben, sind möglich. Wenn Sie nach der VORSCHAU Änderungen am Dokument vornehmen, kann die durch die Vorschau ausgelöste Korrektur nicht mehr rückgängig gemacht werden.

Auswahl oder Dokument

Das Skript bearbeitet entweder den gerade ausgewählten Textbereich oder das komplette Dokument. Die Entscheidung wird davon abhängig gemacht, ob beim Start des Skripts ein Textbereich ausgewählt ist.

Das vorgefertigte Skript soll als Grundlage für eigene Anpassungen dienen. Es enthält die folgenden interaktiven Funktionen:

- Tabulatorzeichen können durch einen Leerraum ersetzt werden.
- Trennstriche mit folgendem Umbruch können entfernt werden.
- Mit Bindestrich getrennte Wörter können kontrolliert werden.
- Harte Zeilenumbrüche, die in Absätzen stehen, deren Absatzformat nicht `Titel` enthält, können zu einem Leerraum korrigiert werden.

Außerdem werden immer die folgenden Korrekturen ausgeführt:

- Leerraum zu Beginn eines Absatzes wird entfernt.
- Durch die Formatoption BUCHSTABENART → GROSSBUCHSTABEN erstellte Großbuchstaben werden zu echten Großbuchstaben konvertiert.

Das Skript erweitern

Bevor man das Skript produktiv einsetzt, sollten die Anforderungen für das konkrete Projekt geklärt sein. Beispielsweise ist die Voraussetzung, dass die Namen aller Überschriften-Absatzformate den String `titel` enthalten, in der Praxis nicht immer gegeben.

Im Folgenden wird nicht das ganze Skript, sondern nur die Anpassung der Suchen/Ersetzen-Funktion und das Einbinden eigener Funktionen zur Korrektur beschrieben. Die Konfiguration des Skripts wird mit dem Objekt `fcList` ganz zu Beginn des Skripts vorgenommen.

Listing 135 Das Konfigurationsobjekt

```
var fcList = {
  iaGREP : [
    {findGREP:"-[\\n\\r]", changeString:"",
     errorText:"Trennung + Umbruch?",
     solutionText:"Trennstrich und Umbruch werden entfernt"},
    {findGREP:"(?<=[\\l\\u\\d])-(?=[\\l\\d])", changeString:"",
     errorText:"Unklare Trennung?",
     solutionText:"Trennstrich wird entfernt"}
  ],
  defaultGREP : [
    {findGREP:"^\\h+", changeGREP:""},
  ],
  iaFunctions : [
    {findFunction:"harterZeilenumbruchAusserTitel",
     changeFunction:"fix_harterZeilenumbruchAusserTitel",
     errorText:"Harter Zeilenumbruch (Ohne AF die \"titel\"
     enthalten)",
     solutionText:"Zu Leerraum ersetzen"},
  ],
  defaultFunctions : [
    {findChangeFunction:"echteGrossbuchstabenErstellen"},
  ]
}
```

Funktion des Skripts

Im Konfigurationsobjekt `fcList` können GREP-Suchen sowie selber erstellte Funktionen eingetragen werden. Grundsätzlich wird unterschieden zwischen Anweisungen, die vom Benutzer interaktiv gelöst werden müssen (`ia...`), und Anweisungen, die immer ausgeführt werden (`default...`). Die Eigenschaften `iaGREP` und `defaultGREP` nehmen Suchen/Ersetzen-Anweisungen auf. In `iaFunctions` können Suchen/Ersetzen-Funktionen eingetragen werden. In `defaultFunctions` werden Korrekturfunktionen eingetragen, die ein Problem selbstständig lösen. Die vier Eigenschaften des Objekts `fcList` enthalten jeweils einen Array. Informationen zur Erstellung von eigenen Objekten finden Sie auf Seite 121.

Im Beispiel-Code werden in `iaGREP` verschiedene Arten von fehlerhaften Trennungen lokalisiert. In `iaFunctions` werden harte Zeilenumbrüche außerhalb von Überschriften über eine eigene Korrektur-

funktion lokalisiert. Automatisch werden in `defaultGREP` fehlerhafte Leerräume gelöscht und in `defaultFunctions` falsch formatierter Text in echte Großbuchstaben konvertiert.

Interaktive Suchen/ Ersetzen-Anweisungen

Der Array in der Eigenschaft `iaGREP` enthält Elemente, die jeweils eine Suchen/Ersetzen-Anweisung und deren Beschreibung beinhalten. Die Eigenschaft `findGREP` muss eine GREP-Suchanfrage für den zu suchenden Fehler enthalten. Im Beispielskript werden zwei Probleme gesucht. Mit `-[\n\r]` wird nach einem Bindestrich, gefolgt von einem harten Zeilenumbruch oder Absatzende gesucht. Möglicherweise fehlerhafte Trennungen werden mit `(?<=[\l\u\d])-(?=[\l\d])` lokalisiert. Es werden Bindestriche zwischen einem beliebigen Buchstaben oder einer Zahl und einem Kleinbuchstaben oder einer Zahl gefunden.

Für die GREP-Suchen in diesem Skript ist der Einsatz von Look Around Assertions (Unterkapitel 10.6) wichtig, weil die Fehlerstelle nur die eigentlich fehlerhaften Zeichen enthalten darf. Der GREP muss als String übergeben werden, entsprechend muss der Backslash mit `\\` doppelt maskiert werden.

Die Eigenschaft `changeString` enthält einen String für das Korrekturzeichen. Wenn die gefundene Fehlerstelle gelöscht werden soll, übergibt man einen leeren String. Sie dürfen hier keinen GREP verwenden, weil das Skript nur mit festen Werten korrigieren kann. Der Text der Eigenschaft `errorText` wird als Beschreibung im Dialog angezeigt, der Text der Eigenschaft `solutionText` wird als Hinweis beim Button KORREKTUR angezeigt. Sie können hier weitere Objekte mit eigenen Suchanfragen hinzufügen, vergessen Sie dabei nicht das Komma zwischen den Objekten.

Default Suchen/ Ersetzen-Anweisungen

In der Eigenschaft `defaultGREP` ist ein Array mit Objekten gespeichert, die eine Suchen/Ersetzen-Anweisung per GREP steuern. Die Objekte sind ähnlich zu den Objekten in `iaGREP` aufgebaut. Sie enthalten die Eigenschaft `findGREP` für die Suche und `changeGREP` für die Ersetzung. In `changeGREP` kann, wie der Name vermuten lässt, auch eine GREP-Ersetzung mit Zugriff auf die Fundstelle eingetragen werden. Im Standardskript werden mit der Suche nach `^\h+` alle Leerräume zu Beginn eines Absatzes entfernt. Diese Leerräume sind immer Fehler und können ohne Benutzerinteraktion gelöscht werden. Falls Sie InDesign-Spezialzeichen wie Indexmarken verwenden, beachten Sie bitte auch das Unterkapitel 10.16 zu diesem Thema.

Interaktive Suchen/ Ersetzen-Funktionen

Die Eigenschaft `iaFunctions` ist recht ähnlich aufgebaut. Zur Suche und Korrektur werden aber in den Eigenschaften `findFunction` und `changeFunction` die Namen von selbst erstellten Funktionen übergeben. Das hat den Vorteil, dass innerhalb dieser Funktionen alle InDesign-Funktionen angesteuert werden können und mit Hilfe von

if-Abfragen die Steuerung verfeinert werden kann. Im Beispielskript wird die Funktion `harterZeilenumbruchAusserTitel` für die Suche und `fix_harterZeilenumbruchAusserTitel` eingesetzt. Die beiden Funktionen werden hier exemplarisch vorgestellt; wenn Sie eigene Funktionen hinzufügen wollen, müssen Sie diese nach dem gleichen Schema erstellen.

Listing 136 `harterZeilenumbruchAusserTitel()`

```
function harterZeilenumbruchAusserTitel (context) {
  if (app.findChangeGrepOptions.hasOwnProperty ("searchBackwards")){
    app.findChangeGrepOptions.searchBackwards = false;
  }
  app.findGrepPreferences = NothingEnum.NOTHING;
  app.changeGrepPreferences = NothingEnum.NOTHING;
  app.findGrepPreferences.findWhat = "\\n";
  var results = context.findGrep(true);
  for (var i = results.length - 1; i >= 0; i--) {
    result = results[i];
    if (result.appliedParagraphStyle.name.toLowerCase().
    indexOf("titel") > -1) {
      results.splice (i, 1);
    }
  }
  app.findGrepPreferences = NothingEnum.NOTHING;
  app.changeGrepPreferences = NothingEnum.NOTHING;
  return results;
}
```

Aufbau der Suchfunktion

Die Funktion für die Suche übernimmt einen Parameter, im Beispiel `context`, der den Suchbereich enthält – das kann ein Objekt vom Typ `Document` oder `Text` sein. Für diese Funktion spielt das keine Rolle, weil beide Objekte die Methode `findGrep()` enthalten. Wenn Sie eigene Funktionen entwickeln und auf Eigenschaften oder Methoden zugreifen, die nicht in beiden Objekten enthalten sind, müssen Sie den Typ des Objekts vorab bestimmen und darauf reagieren (→ Seite 162). Die Funktion muss einen Array mit Textstellen (den Fehlern) zurückliefern – im Beispiel sind die Fehler in `results` enthalten. Dieser Array wird dann im Dialog abgearbeitet.

Die Funktion enthält die Suche nach einem GREP, deren Ergebnis in einer for-Schleife geprüft und verfeinert wird. Die Idee ist, dass alle harten Zeilenumbrüche, die innerhalb von Überschriften stehen, für den EPUB-Export erhalten bleiben sollen und deswegen auch nicht als Fehler gemeldet werden müssen. In der Praxis muss die Namensprüfung an die tatsächlichen Gegebenheiten angepasst werden.

Das Vorgehen ist ähnlich dem FindAndDo-Skript aus Unterkapitel 4.9. Innerhalb der for-Schleife wird allerdings keine Ersetzung vorgenommen, sondern es wird geprüft, ob der Name des Absatzformats den String `titel` enthält. Wenn dies der Fall ist, steht der harte Zeilenumbruch in einem Titel und soll nicht entfernt werden. Die Textstelle

muss aus dem Array mit den Fehlern entfernt werden. Dazu wird die Array-Funktion `splice()` eingesetzt, als erster Parameter wird der Index des Elements, als zweiter Parameter die Anzahl der zu entfernenden Elemente übergeben. Nach der Prüfung wird der bereinigte Array mit `return` zurückgeliefert.

Listing 137 `fix_harterZeilenumbruchAusserTitel()`

```
1  function fix_harterZeilenumbruchAusserTitel (args) {
2    var currentError = args[0];
3    currentError.contents = " ";
4  }
```

Aufbau der Ersetzungsfunktionen

Die Funktion erhält als Parameter einen Array, der als einziges Element ein Objekt vom Typ `Text` an der Stelle des Fehlers enthält. Der Zugriff auf das Textelement in der zweiten Zeile der Funktion muss also so oder ähnlich in allen Ersetzungsfunktionen durchgeführt werden. Der Grund für die Parameterübergabe ist der Aufruf der Funktion im Hauptskript.

Die Korrektur an dieser Stelle ist einfach, es wird der Inhalt der Textstelle mit einem Leerzeichen ersetzt. In der Korrekturfunktion kann aber ausgehend von der Textstelle beliebig programmiert werden. Denkbar ist das Zuweisen von Formaten, beispielsweise beim Umbau von Aufzählungen, oder die Korrektur von Abständen, die durch leere Absätze erzeugt wurden.

Default Korrekturfunktionen

In der Eigenschaft `defaultFunctions` werden Objekte gesammelt, die Funktionen beschreiben, die generell ausgeführt werden. Es wird nur der Funktionsname in der Eigenschaft `findChangeFunction` benötigt. Die Funktion muss im Skript enthalten sein und sich um die Korrektur kümmern. Als Parameter wird das Objekt, in dem die Ersetzungen vorgenommen werden sollen, übergeben. Die Funktion `echteGrossbuchstabenErstellen()` aus dem Beispielskript wandelt mit der Funktion Buchstabenart → Grossbuchstaben erstellte Großbuchstaben in echte Großbuchstaben und kann exemplarisch für die Erstellung von Korrekturfunktionen analysiert werden. Eine ähnliche Funktion könnte für die Konvertierung von Buchstabenart → Kapitälchen geschrieben werden.

Technische Details zum Skript

Session

Das Skript arbeitet mit einer ScriptUI-Palette (→ Seite 205). Paletten sind nicht modal, so dass Sie auch im Dokument Änderungen vornehmen können, während die Palette angezeigt wird. Damit die Palette auch nach dem Ende des Skripts noch erreichbar ist, wird zu Beginn des Skripts mit `#targetengine` eine Session angelegt (→ Seite 206).

Globales Objekt

Um später die Funktionen aufzurufen, wird das Objekt `global` benötigt. Es repräsentiert den globalen Bereich und wird normalerweise implizit verwendet; wenn Sie `app` oder `alert()` verwenden, greifen Sie

InDesign automatisieren

Gregor Fellenz studierte nach seiner Ausbildung zum Mediengestalter an der Hochschule der Medien in Stuttgart Druck- und Medientechnik. Während des Studiums konzentrierte er sich auf die Themen XML und Cross-Media-Publishing.

Seit 2004 beschäftigt er sich beruflich mit Publishing-Workflows für Print und digitale Medien. Sein Schwerpunkt liegt auf der automatisierten Erstellung von Publikationen mit InDesign und XML. Neben der Implementierung von Publishing-Workflows ist er als Berater, Trainer und Projektleiter tätig. Unter @grefel twittert er über die aktuellen Entwicklungen im Bereich InDesign-Skripting.

Gregor Fellenz

InDesign automatisieren

Keine Angst vor Skripting, GREP & Co.

2., aktualisierte und erweiterte Auflage

dpunkt.verlag

Lektorat: Barbara Lauer, Bonn
Copy-Editing: Alexander Reischert (Redaktion Aluan, Köln)
Satz: Bidirektionaler XML-Workflow, Gregor Fellenz
Herstellung: Susanne Bröckelmann
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: Media-Print Informationstechnologie, Paderborn
Bildnachweis: Nora Klein: Seite 2; Teresa Wurtz: Seite 295; Gregor Fellenz: alle weiteren.

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN 978-3-86490-235-2

2. Auflage 2015
Copyright © 2015 dpunkt.verlag GmbH
Wieblinger Weg 17
69123 Heidelberg

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.
Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.
Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.
5 4 3 2 1 0

Papier plus PDF.

Zu diesem Buch – sowie zu vielen weiteren dpunkt.büchern – können Sie auch das entsprechende E-Book im PDF-Format herunterladen. Werden Sie dazu einfach Mitglied bei dpunkt.plus+:

www.dpunkt.de/plus

Inhalt

Einleitung

Vorwort zur 2. Auflage

Nachdem sich Anfang 2014 abzeichnete, dass die erste Auflage bald ausverkauft sein würde, sagte ich zu meiner Lektorin Barbara Lauer, dass ein Update schnell gemacht wäre. Knapp ein Jahr später weiß ich, dass diese Einschätzung falsch war. Auch wenn der bewährte Aufbau des Buches grundlegend gleich geblieben ist, sind nur wenige Seiten von einer Überarbeitung gänzlich verschont worden.

Das hat verschiedene Gründe: Zunächst natürlich viele Neuigkeiten und Erleichterungen in InDesign CS6 und CC – alle Referenzen auf ältere Versionen habe ich entfernt, damit das Buch übersichtlich bleibt. Aber auch Erfahrungen aus Schulungen, Gesprächen und Feedback zur ersten Auflage wollten eingearbeitet werden.

Besonders zu erwähnen ist das aus meinem Schulungskonzept übernommene Beispielskript, das die Einführung in JavaScript in Kapitel 6 begleitet. Einige Kochrezepte sind entfallen, dafür neue Beispiele aus meiner täglichen Praxis hinzugekommen – erwähnenswert ist der Umgang mit CSV, die Realisierung von Stapelverarbeitungen sowie das Verketten von GREP-Abfragen. Außerdem wurde das Buch mit Beispielen zum Skripting der interaktiven Funktionen erweitert.

Im Bereich E-Book hat sich so viel verändert, dass das komplette Kapitel ausgetauscht werden musste. Hier habe ich den Fokus auf das Skripting des Exports gelegt, die eigentliche Erstellung von E-Books musste entfallen – hier gibt es inzwischen viele andere gute Quellen.

Mehr aus InDesign herausholen

Adobe InDesign hat sich zu einem der wichtigsten Desktop-Publishing-Programme entwickelt und ist aus der professionellen Gestaltung und Produktion von Druckerzeugnissen kaum mehr wegzudenken. Die meisten Anwender verwenden allerdings nur die offensichtlichen und gebräuchlichsten Funktionen und Features. Viele Möglichkeiten gehen in der Fülle der Funktionen unter oder es bleibt schlicht keine Zeit, sich während des Tagesgeschäfts damit auseinanderzusetzen. Selbst gestandene InDesign-Profis lernen immer wieder neue Arbeitsweisen hinzu.

Die Produkte, die mit InDesign erstellt werden, sind außerordentlich vielfältig. Die Bereiche der professionellen Anwendung gehen von der klassischen Druckvorstufe über die Erstellung von digitalen Medien bis zu serverbasierten Web2Print-Lösungen. Entsprechend vielseitig sind auch die Szenarien und Workflows, in denen InDesign eingesetzt wird.

Wer sollte InDesign automatisieren?

Eine Automatisierung lohnt sich meist nur, wenn wiederkehrende Aufgaben zu lösen oder große Datenmengen zu verarbeiten sind. Klassische Beispiele sind die Katalogproduktion, technische Dokumentationen oder die Buchherstellung. Aber auch die Erstellung von Zeitschriften, Magazinen, Broschüren sowie ganz allgemein von Periodika lässt sich durch Automatisierung gut unterstützen. Im Bereich der digitalen Medien können sowohl E-Books als auch digitale Magazine mit Hilfe von Skripting effizienter erstellt werden.

Es liegt in der Natur der Sache, dass sich das kreative Gestalten eines neuen Corporate Designs, einer Werbeanzeige oder Ähnlichem kaum automatisieren lässt. Doch selbst in diesem Bereich lassen sich mit Skripting-Kenntnissen Variationen einer Gestaltung erstellen, beispielsweise kann man generell alle Schriftgrößen um einen bestimmten Prozentwert verringern. Denn auch hier gilt: Statt ständig zehnmal klicken, lieber einmal programmieren.

Unabhängig davon kann eine Automatisierung auch die regelbasierte Gestaltung unterstützen, da Ausnahmen einen höheren Aufwand bedeuten und dadurch neu hinterfragt werden. Es muss auch nicht immer eine ausgefeilte Layoutautomatisierung sein – manchmal helfen schon kleine Skripte oder das Wissen über die beeindruckenden Möglichkeiten der Suchen/Ersetzen-Funktion mit GREP weiter.

Suchen und ersetzen mit GREP

GREP bietet unabhängig vom Skripting eine oft übersehene Möglichkeit, Anforderungen, die an Dokumente gestellt werden, schnell und effizient zu lösen. Mit GREP können komplexe Suchanfragen und Ersetzungsanweisungen definiert werden, die den Arbeitsalltag ganz erheblich erleichtern. Dazu werden Reguläre Ausdrücke verwendet, die auch in anderen Programmen zum Einsatz kommen.

Skripting

Eine weitere Möglichkeit ist die Programmierung von InDesign mit selbst erstellten Skripten. Wer »langweilige« oder sich immer wiederholende Arbeiten automatisieren kann, spart viel Zeit, um sich auf die wesentlichen Aufgaben zu konzentrieren. Ich würde sogar sagen, dass die Möglichkeit der Programmierung oder besser gesagt des Skriptings das beste und produktivste Feature von InDesign ist.

Fast alles, was Sie mit der normalen Benutzeroberfläche realisieren können, kann auch über ein Skript erreicht werden. Ob Sie eine neue Seite anlegen, einen Textrahmen aufziehen oder einen Absatz formatieren wollen – per Skript ist das alles kein Problem. Skripte erlauben sogar noch mehr: Mit ihnen kann man eigene Menüs definieren oder auf Benutzeraktionen reagieren.

E-Books und XML

Neben GREP und Skripting wird die Zweitverwertung von gedruckten Inhalten in digitaler Form immer wichtiger. Die hier verwendeten Technologien EPUB für E-Books und XML als Schnittstelle zu strukturierten Datenbeständen oder sogar Datenbanken werden meist mit Hilfe von Skripten optimiert oder überhaupt erst möglich gemacht.

Gerade in diesem Bereich zeigt sich auch die Entwicklung hin zu schnelleren Produktionszyklen und dem damit einhergehenden Zeit- und Kostendruck. Oft müssen die elektronischen Publikationen zeitgleich mit dem Printwerk erhältlich sein. Mit Kenntnissen im Bereich Automatisierung kann man diesen Entwicklungen entspannt entgegensehen.

Welche Versionen werden beschrieben?

Das Skripting von InDesign ist in den Versionen CS6 bis CC 2014 weitgehend identisch. Das hier vorgestellte JavaScript kann plattformunabhängig auf Windows und Mac OS eingesetzt werden.

Wenn sich Unterschiede in der Programmierung ergeben, sind diese im Buch durchgehend gekennzeichnet. Beispieldaten wurden gegebenenfalls für alle notwendigen Versionen erstellt.

An wen richtet sich das Buch?

Jeder kann InDesign automatisieren! Das Buch setzt lediglich solide InDesign-Grundkenntnisse voraus. Erfahrungen in einer Skript- oder Programmiersprache sind von Vorteil, werden aber nicht vorausgesetzt. Ich habe den Anspruch, auch Anfänger auf dem Weg zur InDesign-Automatisierung zu begleiten!

Die Einführung in die Automatisierung kann jedem gelingen. Die Beispiele sind aus dem Arbeitsalltag von Mediengestaltern, Grafikern und Verlagsherstellern entnommen. Aber auch Medieninformatiker oder Webdesigner werden viele praxisorientierte Hinweise für die Produktion von Printmedien finden.

Wer kann skripten?

Viele Anwender schrecken vor der Hürde der Programmierung zurück. Doch liegen gerade darin unermessliche Möglichkeiten, sich das Arbeitsleben einfacher zu machen. Etwas überspitzt könnte man sagen: »Nur ein fauler Mensch macht produktive Skripte.« Denn wer kein Problem damit hat, hundertmal denselben Arbeitsschritt auszuführen, anstatt einen Kaffee zu trinken, der braucht auch keine Automatisierung.

Zugegeben, das erste Mal sind die nackten Codezeilen eher abschreckend und man weiß oft nicht, wo man anfangen soll. Wer sich allerdings auf die Reise begibt, wird bald feststellen, dass das Themengebiet logisch aufgebaut ist und sich mit dem richtigen Einstieg meistern lässt.

Außerdem macht Programmieren Spaß. Das glauben Sie nicht? Sobald Sie Ihren eigenen Skripten bei der Arbeit zuschauen, werden Sie Ihre Meinung ändern. Lassen Sie sich nicht vom ersten Fehler entmutigen – das Verständnis fürs Programmieren entwickelt man leider nicht über Nacht.

JavaScript

In diesem Buch wird das Skripting anhand der Skriptsprache JavaScript erläutert, die sowohl auf Windows- als auch Mac OS-Systemen eingesetzt werden kann. Genauer gesagt wird die erweiterte JavaScript-Implementierung von Adobe mit dem Namen ExtendScript verwendet.

JavaScript wird auch für die Programmierung von Webseiten benutzt. Leider ist nur der Sprachkern identisch, der Umgang mit dem Browser bzw. InDesign aber völlig verschieden.

Danksagung

Ich möchte mich bei allen, die an der Entstehung dieses Buches beteiligt waren, bedanken. Dazu zählt vor allem Rebecca, die das Projekt von der ersten Idee bis zur Endkorrektur unterstützt hat.

Besonders hervorheben möchte ich auch meine Lektorin Barbara Lauer, die mir stets motivierend und konstruktiv zur Seite stand, die Gutachter der ersten Auflage Sarah Schäfer, Nadine Thiele, Marco Morgenthaler, Marko Hedler, Martin Fischer, Gerald Singelmann und Tobias Fischer, die mir viele wichtige Anregungen und Hinweise gegeben haben. Ein Dank gilt auch meinen Eltern, die das komplette Manuskript Korrektur gelesen haben. Für die zweite Auflage kamen wichtige Inspirationen von Klaas Posselt, Kai Rübsamen, Uwe Laubender und Stefan Göbel.

Viele weitere Personen, die Anregungen für einzelne Skripte geliefert haben, werden im Verlauf des Buches genannt.

Verwendung dieses Buchs

Das Buch ist als Praxis- und Lernbuch aufgebaut, das didaktische Konzept beruht auf den Erfahrungen verschiedener Schulungen und meiner Lehrveranstaltung an der Hochschule der Medien in Stuttgart. Alle Skripte werden besprochen und, wann immer möglich, durch Beispieldokumente erläutert. Unabhängig davon kann vor allem der zweite Teil auch als Nachschlagewerk verwendet werden.

Das Buch ist in drei Teile gegliedert. Der erste Teil bietet einen sanften Einstieg in die Programmierung und Automatisierung von InDesign. Der zweite Teil ist eine systematische Einführung in die InDesign-Programmierung mit JavaScript. Der dritte Teil enthält praxisorientierte Kochrezepte für konkrete Problemstellungen.

I. Teil

Der Einstieg über die Suche und Ersetzung mit GREP in **Kapitel 1** bietet eine in sich geschlossene Heranführung an die abstrakte Formulierung von Lösungen, wie sie auch beim Skripting üblich ist. **Kapitel 2** zeigt, wie Skripte in InDesign installiert werden und wie Sie diese produktiv einsetzen können. In **Kapitel 3** wird das ExtendScript Toolkit, das Werkzeug für die Erstellung von Skripten, vorgestellt. In **Kapitel 4**

werden einfache Skripte für alltägliche Praxisprobleme im Detail besprochen. Alle Skripte werden zeilenweise diskutiert und können anhand von Beispieldokumenten ausprobiert werden. Dieses Kapitel enthält alle wichtigen Programmiermethoden, behält aber den Fokus auf einem möglichst sanften Einstieg. An vielen Stellen verweist es auf den systematischen Einstieg im zweiten Teil des Buchs. **Kapitel 5** enthält nützliche Tipps zur Fehlersuche und verrät, wo man Hilfe bekommen kann.

II. Teil

Der zweite Teil beginnt in **Kapitel 6** mit einer systematischen Einführung in die Programmiersprache JavaScript. Hier werden alle wichtigen Programmiermethoden und -konzepte für die InDesign-Automatisierung vorgestellt. Das Kapitel schließt mit einem Exkurs zum Thema Unicode, dem Standard für die Kodierung von Zeichen in Schriften und Dateien. **Kapitel 7** dreht sich um das InDesign-Objektmodell. Dieses Kapitel enthält den Schlüssel zur InDesign-Programmierung, es ist als praxisorientierte Referenz aufgebaut. Zu allen Themen finden Sie wieder dokumentierte Beispielskripte.

In **Kapitel 8** wird das Debugging, das »Fehlerfinden«, mit dem ExtendScript Toolkit besprochen. **Kapitel 9** stellt Programmierkonzepte für die Automatisierung von Printprodukten vor und zeigt Best-Practice-Beispiele für gute Skripte.

III. Teil

Der dritte Teil ist thematisch in fünf Bereiche aufgeteilt. Zunächst wird das Thema GREP in **Kapitel 10** vertieft und mit Praxisbeispielen abgerundet. Hier sei auch auf die GREP-Referenz Anhang A1 verwiesen, die alle Metazeichen für die Suche mit GREP enthält.

Kapitel 11 enthält Rezepte für konkrete Aufgabenstellungen. Sie können die einzelnen Unterkapitel losgelöst voneinander bearbeiten. Für alle Skripte gibt es Beispieldokumente.

In **Kapitel 12** wird ein umfassender Automatisierungs-Workflow mit Word-Dateien vorgestellt. Hier wird eine Word-Datei per Skript in ein nahezu fertig gestaltetes InDesign-Dokument überführt.

Kapitel 13 stellt den Export für E-Books im Format EPUB vor. Der Schwerpunkt liegt auf Skripten für die Optimierung des Exports.

In **Kapitel 14** wird der Umgang von InDesign mit XML vorgestellt. Neben den normalen Programmfunktionen werden auch die Möglichkeiten per Skripting erörtert.

Wegweiser durch das Buch

Für einen umfassenden Einstieg können Sie Teil I und II durchgehend lesen. Alternativ können Sie das Buch auch mit dem Schwerpunkt GREP bzw. Skripting durcharbeiten.

Schwerpunkt GREP

Falls Sie sich zunächst nur für die Suche und Ersetzung mit Regulären Ausdrücken interessieren, ist Kapitel 1 der richtige Einstieg für Sie. Im Unterkapitel 10.1 wird gezeigt, wie man mehrere GREP-Abfragen hintereinander ablaufen lassen kann. Weitergehende GREP-Techniken finden Sie in Kapitel 10, das sich in einen Theorie- (→ Unterkapitel 10.2 bis 10.10) und Praxisteil aufteilt (→ Unterkapitel 10.11 bis 10.16). Eine vollständige GREP-Referenz finden Sie auf Seite 371.

Schnelleinstieg Skripting

Wenn Sie bereits Programmiererfahrung haben und direkt mit dem Skripting beginnen wollen, ist der folgende Fahrplan empfehlenswert: Informieren Sie sich in Unterkapitel 2.1 über Installation und Ausführung von Skripten. Springen Sie dann zu Kapitel 3 und erlernen Sie die Verwendung der Entwicklungsumgebung ExtendScript Toolkit. Nach einem kurzen Abstecher zum Skript *Hallo Welt* im Unterkapitel 4.2 können Sie sich in Teil II systematisch in die Programmierung mit JavaScript einarbeiten. Auf wichtige Techniken, die im ersten Teil vorgestellt wurden, wird im Text verwiesen. Bei Ihrem ersten Skriptfehler springen Sie zurück zu Kapitel 5, das sich mit häufigen Fehlern beschäftigt.

Schnelleinstieg JavaScript-Profi

Wenn Sie schon Erfahrung in JavaScript gesammelt haben, empfehle ich die folgenden Kapitel: Nach der Übersicht über die Installation von Skripten in Unterkapitel 2.1 informieren Sie sich in Kapitel 3 über die IDE ExtendScript Toolkit. Nun können Sie direkt in die Beschreibung des InDesign-Objektmodells in Kapitel 7 einsteigen. Bei Ihrem ersten Skriptfehler springen Sie nochmal zurück zu Kapitel 5, das sich mit häufigen Fehlern beschäftigt. Werfen Sie auch ein Blick auf die Best-Practice-Konzepte in Kapitel 9.

Webseite zum Buch

Für dieses Buch habe ich die Webseite **http://www.indesignjs.de** eingerichtet. Hier finden Sie alle digitalen Informationen zum Buch sowie aktuelle Neuigkeiten. Über meinen Twitter-Account *@grefel* veröffentliche ich außerdem aktuelle Informationen zum Buch und zur InDesign-Automatisierung

Linkverkürzer

Da niemand zeilenlange Internetadressen aus einem Buch abtippen möchte, habe ich für die meisten Adressen einen Linkverkürzer eingerichtet. Sie erkennen diese Links an dem ↗ Symbol. Die darauffolgende Zahl müssen Sie mit der Adresse *indesignjs.de* kombinieren. Für ↗18 müssen Sie also http://www.indesignjs.de/18 in die Adresszeile des Browsers eingeben.

Skripte und Beispieldaten

Mit Download ZIP alle Daten als ZIP-Archiv herunterladen

Ich habe alle Skripte und die Beispieldaten, die im Buch erwähnt werden, auf GitHub zur Verfügung gestellt. Sie können die Daten unter https://github.com/grefel/indesignjs ↗2 herunterladen.

Schriften
In allen Beispieldokumenten werden die Schriften *Alegreya* und *Alegreya Sans* von Juan Pablo del Peral verwendet 139. Sie können bei www.fontsquirrel.com 140 kostenfrei heruntergeladen werden.

InDesign-Skripting-Kurzreferenz
Zum Buch gehört eine InDesign-Skripting-Kurzreferenz mit einer Übersicht der wichtigsten Objekte, Eigenschaften und Methoden des InDesign-Objektmodells. Sie können ein PDF unter http://www.indesignjs.de/idskurzreferenz.pdf 1 herunterladen.

... und außerdem
Hinweis
Die notwendige Reduktion der Komplexität führt leider manchmal dazu, dass die Skripte nur in den beschriebenen Situationen funktionieren. Um unerwünschte Nebeneffekte auszuschließen, sollten Sie die Skripte nur dann in einem produktiven Umfeld einsetzen, wenn Sie verstanden haben, wie diese genau funktionieren. Spätestens am Ende des ersten Teils sollten Sie die Fähigkeit erlangt haben, die Wirkungen und Nebenwirkungen von Skripten zu verstehen, aber vor allem können Sie dann die Skripte an Ihre eigenen Anforderungen anpassen!

Exkurse
In vielen Fällen kommt man bei der Automatisierung mit Technologien in Berührung, über die man ein eigenes Buch schreiben könnte. Damit das Buch auch für Leser ohne Vorkenntnisse lesbar bleibt, habe ich an diesen Stellen kurze Exkurse eingefügt. Diese bieten nur einen minimalen Einstieg und ersetzen nicht das Studium der Technologie. Im Anhang finden Sie Internetressourcen und Buchempfehlungen.

Experten
Einige Themen und Konzepte richten sich an fortgeschrittene Anwender. Diese Bereiche sind wie dieser Absatz hervorgehoben und in der Marginalspalte gekennzeichnet.

Errata
Das Buch ist sicher nicht umfassend und trotz vieler Tests nicht fehlerfrei. Eventuelle Fehler werden auf http://www.indesignjs.de im Bereich *Errata* veröffentlicht.

Zu den vorgeschlagenen Lösungen gibt es sicher auch Alternativen, die im Rahmen dieses Buches nicht alle dargestellt werden können. Gerne veröffentliche ich interessante Lösungsalternativen auf der Webseite zum Buch. Schicken Sie Ihre Ideen mit einem Hinweis, dass diese zur Veröffentlichung gedacht sind, an *gregor.fellenz@publishingx.de*. Sie werden zusammen mit weiteren Themen im Bereich *Neuigkeiten* veröffentlicht.

Tastenkombinationen

Die geläufigen Tastenkombinationen von InDesign setze ich voraus, besonders wichtige oder praktische erwähne ich im Text. Die meisten Tastenkombinationen von InDesign sind unter Windows und Mac identisch. Für die Taste STRG unter Windows wird unter Mac OS CMD bzw. ⌘ verwendet. Im Buch fasse ich die beiden Tasten mit BEFEHL zusammen.

Formatierungen und deren Bedeutung

Hervorhebungen, wichtige Begriffe, Konzepte oder Techniken sowie Dateinamen und Pfade	*Kursiv*
Navigation zu Befehlen über InDesign-Menüs und Shortcuts sowie Bezeichnungen von InDesign-Dialogen, Bedienfeldern und Menüs	DATEI → NEU
Suchanfragen und Ersetzungsanweisungen für die Arbeit mit GREP	`Such- oder Ersetzungstext`
Code oder Codebestandteile, die im Text erwähnt werden	`Code`
Hervorhebungen im Code	**`Fett`**
Hinweise auf wichtige Themen in der Marginalspalte	!¡
Hinweis auf fortgeschrittene Themen in der Marginalspalte	++
Linkverkürzer, die Zahl in Kombination mit der Adresse http://www.indesignjs.de führt Sie zur Webseite.	18

Feedback

Ich freue mich über jedes Feedback, Kritik oder Verbesserungsvorschläge und natürlich auch über Fehlermeldungen. Sie können mich unter *gregor.fellenz@publishingx.de* erreichen.