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.

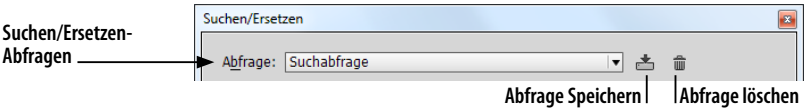


Abb. 35
Suchen/Ersetzen-Abfragen speichern

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.

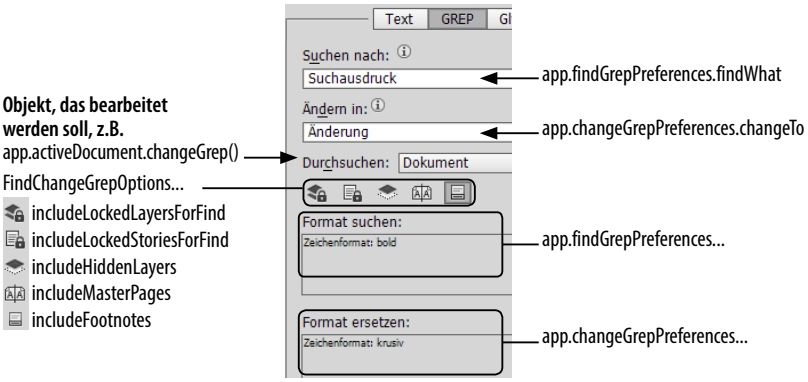
Listing 21
4-8_SuchenUnd Ersetzen-2.jsx

```
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.

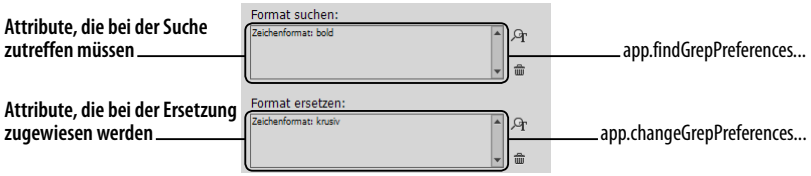


Abb. 37
Änderung der Formatangaben im Suchen/Ersetzen-Dialog

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.