Technologisches Praktikum InDesign Satzautomation

11740 TP: Cross-Media-Publishing

7., 14., 21. und 28. November 2009
10:00 Uhr bis 17:00 Uhr
Raum 171

Gregor Fellenz

Mediengestalter für Digital- und Printmedien
Studium Druck- und Medientechnologie, HdM Stuttgart
pagina GmbH, Tübingen

gregor.fellenz@indesignblog.com

Übersicht

1. InDesign Programmierung mit JavaScript

- Einführung JavaScript
- InDesign Objektmodell
- InDesign Programmierung
- Programmierkonzepte Layoutautomation

2. InDesign und XML

- Zusammenspiel von strukturierten Daten und Satzautomation
- XML in InDesign bearbeiten
- InDesign XML-Programmierung mit XML-Rules
- XML-Roundtripping

Die Folien zum Download:

http://www.indesignblog.com/hdm

Literatur, Informationen und Quellen

JavaScript

- SelfHTML
  http://de.selfhtml.org/javascript/
  Weblastig, JavaScript Einsteigerseite, leider nicht so gut wie der HTML Teil.
- Core JavaScript 1.5 Reference
  https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference
  Wer's theoretisch mag…
- JavaScript: The Definitive Guide, Fifth Edition, David Flanagan
  http://oreilly.com/catalog/9780596101992/index.html
- JavaScript: The Good Parts, Douglas Crockford
  http://oreilly.com/catalog/9780596517748/index.html
  Anspruchsvolle Übersicht zu Konzepten und zum Einsatz von JavaScript

InDesign und Scripting

- InDesign mit JavaScript automatisieren, Peter Kahrel
  http://www.oreilly.de/catalog/indesignjsger/index.html
  Einziges Buch, Systematische Einführung
- Adobe Scripting Guide
  http://www.adobe.com/products/indesign/scripting/
- **Adobe Scripting Forum** (sehr rege Community)
  http://forums.adobe.com/community/indesign/indesign_scripting
- **Hilf dir Selbst Forum**
  http://www.hilfdirselbst.ch/foren/Adobe_InDesign_Forum_4.html
- Adobe Informationen zu XML
  http://wwwimages.adobe.com/www.adobe.com/products/indesign/scripting/pdfs/indesign_and_xml_technical_reference.pdf

Ziele – oder wozu das alles?

In jedem Layoutprozess fallen routinemäßige Aufgaben an, die sich zwar durch Handarbeit lösen lassen, aber in der Ausführung zeit- und nervenaufreibend sind. Skripte können viele dieser Aufgaben übernehmen.

- **Fertige Scripts** einsetzen oder minimal anpassen...
  Reichhaltige Auswahl von Adobe Skripts oder im Netz
- **Kleine Helferlein** widerkehrende Aufgaben lösen.
  Statt 10 mal klicken einmal programmieren. Verweise auflösen, Sanftes spationieren
- **Dokumentanalyse**
  Preflight selber programmieren
- **Layoutautomatisierung** aus strukturierten Daten
  Fertige oder fast fertige Layouts erstellen
- Layouts aus **XML-Daten** steuern

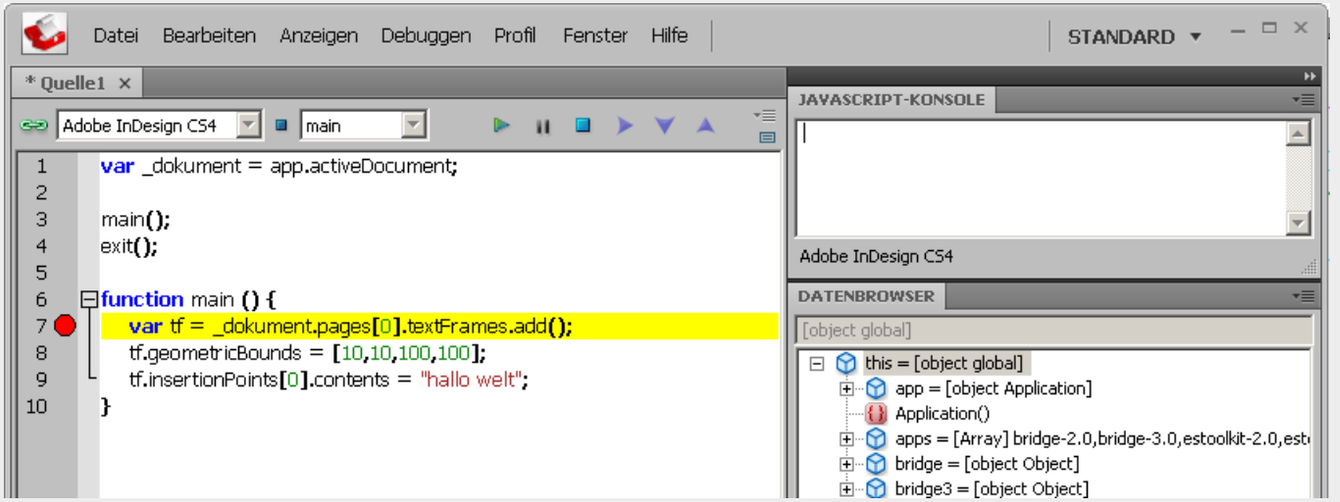
Scripting in InDesign: VB Skript, AppleScript und JavaScript

Einführung JavaScript

- **Plattformunabhängig** (Mac/Windows/[Linux])
- Sprache ist aus der **Web-Programmierung** bekannt.
  Clienstseitige Schnittstelle (API) für den Zugriff auf HTML-Dokumente
  Browser Sandbox aus Sicherheitsgründen
- Kombination aus **Sprachkern** und **Document Object Model** (DOM)
  Wir lernen beides zusammen, der Sprachkern ist auch im Web einsetzbar.
- **Leichte** und **schlanke** Sprache
  Wer schon mal programmiert hat oder informatisches Verständnis mitbringt lernt JS schnell.
- Bei InDesign meist prozedurale Programmierung – es wird aber auf Objekte zugegriffen.
  Layouts/Bücher sind prozedural, haben eine definierte Abfolge und Ende.
- JavaScript in InDesign kann auch auf das **Dateisystem** zugreifen
  Adobe Implementierung

Werkzeuge und Hilfsmittel

- **Texteditor** ausreichend
- **Adobe ExtendedScript Toolkit**
  Editor/Mini **IDE** (Integrierte Entwicklungsumgebung) für InDesign
  Ideal zum Entwickeln von Skripten, Syntaxhervorhebung
- Befehlszeile, **Konsole** zum testen
- Debugging
  Einzelschritte und **Breakpoints**
- Hilfe mit Objektmodell



Kann‘s endlich mal losgehen?

JavaScript-Kochrezept:

1. **InDesign** und **Adobe ExtendScript** starten
2. Programm schreiben
3. Entweder direkt aus der IDE oder in der InDesing Skriptpalette ausführen
4. Fertig!

- **Speicherort**: Über das Skript Panel am einfachsten zu finden.
  Skripte InDesign bekannt machen: In folgendes Verzeichnis legen

  Win C:\Dokumente und Einstellungen\[Benutzername]\Anwendungsdaten\Adobe\InDesign\Version 5.0\Scripts\Scripts Panel

  Mac ~/Users/[Benutzername]/Library/Preferences/Adobe InDesign/Version 5.0/Scripts/ScriptsPanel

- **Musterskript** `helloWorld_01.jsx` im Ordner `01_jsx`

```
var _dokument = app.documents.add();
var _tf = _dokument.pages[0].textFrames.add();
_tf.geometricBounds = [20,20,100,100];
_tf.insertionPoints[0].contents = "hallo welt";
```

Variablen

- Eine Variable ist ein **Platzhalter**, Speicherplatz, Referenz, „Lesezeichen“
- Der **Wert** einer Variablen kann sich während des Ablaufs **ändern**
  An verschiedenen Stellen des Programms kann sie ganz unterschiedliche Werte haben.
- Variablen sollten durch das Schlüsselwort **var** deklariert werden
- Einer Variable kann ein Wert zugewiesen werden, dazu dient der Zuweisungsoperator
- `var _variablenName = Wert`
- Case-sensitiv, d.h. **zahl** und **Zahl** sind zwei verschiedene Variablen
  Tipp: Vermeiden Sie Umlaute und Sonderzeichen in Variablennamen.
- Kann beliebige Werte aufnehmen, übliche Unterscheidungen:
  - **Zahlen** (Kommazahlen immer mit Punkt abtrennen z.B. 12.05)
  - **Strings** (Texte in Anführungszeichen z.B. „ein Text“)
  - **Boolesche Werte** (Wahr oder falsch, true bzw. false),
  - **Objekte** (Ein InDesign Dokument, Array)
  - **null** (Schlüsselwort für einen Nullwert, noch kein Wert zugewiesen,
    nicht zu verwechseln mit der Zahl „0“)
- Aber: In JavaScript **nicht typisiert**

Arithmetische Operatoren

JavaScript kann rechnen, dazu stehen die folgenden **Rechenoperatoren** zur Verfügung:

+	Addition (und Verbinden von Strings)
–	Subtraktion
*	Multiplikation
/	Division
%	Modulo (Restrechnung)
++	Erhöhung (Inkrement)
––	Minderung (Dekrement)

Mit Variablen umgehen

- Die Dateien befinden sich im Ordner `01_jsx`
- Öffnen Sie die Datei `helloWorld_02.jsx` im Extended Script Toolkit

! Verwenden Sie für den Text "hallo Welt" eine Variable.

! Geben Sie anstatt von Text Zahlen im Textrahmen aus.

! Verwenden Sie die JavaScript Konsole um Variablen zu analysieren.

! Durch den **Zuweisungsoperator** „=“ wird der Ausdruck rechts vom Gleichheitszeichen ausgewertet und dann dem links stehenden Ausdruck zugewiesen (In diesem Fall liest man also von rechts nach links!).
Berechnen Sie die Summe zweier Zahlen und geben Sie das Ergebnis im Textrahmen aus.

! Was passiert wenn Sie Text und Zahl addieren?

Texte

Text muss innerhalb von JavaScript in Anführungszeichen "" stehen. Texte werden in der Informatik **Strings** genannt.

- Ein String ist eine Zeichenkette, also ein Folge von zusammen gespeicherten Zeichen.
- Steht ein + zwischen zwei Strings werden diese dadurch zu einem Gesamtstring verbunden.
- Sonderzeichen in Strings werden mit dem **Escape Zeichen** eingeleitet

  \n **Neue Zeile** Soft Return
  \r **Neue Zeile** Return
  \t **Tabulator**
  \" **Anführungszeichen** für die Verwendung von Anführungszeichen innerhalb des Texts
  \' **Einfaches Anführungszeichen**
  \\ **Backslash.** Um einen \ auszugeben muss z.B. "c:\\pfad\\datei.txt" geschrieben werden. Ein einfacher Backslash wird als Beginn eines Steuerzeichens interpretiert.

- **Strings können mehr!**
  Sie haben Funktionen und Eigenschaften:
  `String.length` gibt z.B. die Länge der Zeichenkette zurück
  `String.substring(0,1)` gibt das erste Zeichen zurück

Zeichenketten

- Die Dateien befinden sich im Ordner `01_jsx`
- Öffnen Sie die Datei `helloWorld_02.jsx` im Extended Script Toolkit
- Verwenden Sie den Objektmodell Viewer zur Übersicht über die Funktionen von Strings.

! Die folgenden String-Methoden sind im Allgemeinen besonders nützlich, machen Sie sich mit Ihnen vertraut:
`indexOf(...)`, `lastIndexOf(...)` und `substring(...)`
Beachten Sie, dass Positionen bei Null beginnen!

! Kürzen Sie den String `"hallo welt!"` in `"welt!"` und geben Sie den Text aus.

! Strings können mit dem + Operator verknüpft werden.
Fügen Sie die Strings `"hallo" und "welt!" zusammen.`

! Fügen Sie einen String mit Zeilenschaltung zusammen.

! Wenn `var a` den String `"4"` speichert und `var b` den String `"9"` ist das Ergebnis der Addition der Strings `"49"` nicht die Zahl 13.
Mit der Funktion `Number("123")` können Strings in Zahlen verwandelt werden,
Mit der Funktion `(123).toString()` können Zahlen in Strings verwandelt werden
Korrigieren Sie `casting_01.jsx`

Anweisungen durch Abfragen

Während des Programmablaufs können **Abfragen** ausgewertet werden. Dazu wird eine Kontrollstruktur benötigt, die zur Formulierung von Bedingungen dient. Damit können **Verzweigungen** im Ablauf abgebildet werden.

- Bedingte **if-Anweisungen** werden verwendet, um einen Zustand zu prüfen
- Klassisch ist die Abfrage **If Else** mit zwei Anweisungsblöcken
  also: Wenn eine Bedingung **zutrifft** wird der erste Anweisungsblock ausgeführt (if)
  Wenn die Bedingung **nicht zutifft** wird der zweite Anweisungsblock ausgeführt (else), der zweite Block ist optional

1. Prüfung:

```
if( app.selection.length == 0 ) {
  alert("Es ist nichts ausgewählt");
}
```

2. Verzweigung durch Alternative:

```
if( app.selection.length == 0 ) { ... }
else {
  alert("Es ist etwas ausgewählt");
}
```

Wahr oder falsch?

Die Werte **true** und **false** kommen aus der Aussagenlogik. Die Werte sind logischerweise jeweils immer genau richtig oder immer falsch.

- In JavaScript sind die Bezeichner `true` und `false` reserviert, d.h. sie dürfen nur im eigentlichen Sinne verwendet werden und z.B. nicht als Variablennamen.
- Die if Abfrage testet immer, ob der **Ausdruck** (Vergleich/Frage) mit true beantwortet wurde.
  ```
  if (true) { //wird immer ausgeführt }
  ```
- true entspricht dem Zahlenwert 1
  false entspricht dem Zahlenwert 0
  Das verwirrende Beispiel:
  ```
  false == 0 // Ergebnis: true
  ```
- Der zugehörige Datentyp heißt **Boolean** und kann auch in Variablen verwendet werden.

Vergleichsoperatoren für Abfragen

Es gibt sechs **Vergleichsoperatoren** um **Bedingungen** (Ausdrücke) zu formulieren:

- == gleich **Achtung:** Ein Zuweisung **x=11** wird mit einem = vorgenommen.
  Der Vergleichsoperator für Abfragen **if(x = = 11)** besteht aber aus zwei = = .
- != ungleich
- > größer
- < kleiner
- >= größer oder gleich
- <= kleiner oder gleich

Logische Operatoren

Komplexe Abfragen können mit **logischen Operatoren** verknüpft werden:

- && bedeutet logisch und
- || bedeutet logisch oder
- ! bedeutet nicht (Umkehrung)

Abfragen

- Die Dateien befinden sich im Ordner `01_jsx`
- Öffnen Sie die Datei `abfragen.jsx` im Extended Script Toolkit

! Die Variable `_seitenAnzahl` ist mit der Länge der Seiten im aktuell geöffneten Dokument belegt. Geben Sie alternative Texte in Abhängigkeit von der Dokumentlänge aus.
Testen Sie Ihre Kontrollstruktur, indem Sie dem Dokument Seiten hinzufügen.

! Spielen Sie mit den Vergleichoperatoren. Fragen Sie z.B. eine Mindestseitenanzahl ab.

! Testen Sie zwei Texte auf Gleichheit – gibt es einen Unterschied?
Was testet der „>“ bzw. „<“ Vergleichsoperator wenn er auf Strings angewendet wird?

! Definieren Sie Variablen mit den reservierten Wörtern `true` und `false` und verwenden Sie diese in Abfragen.

! Mit dem Vergleichsoperator == können auch Objektzustände abgefragt werden.
Wenn keine Dokument geöffnet ist, enthält das Objekt `app.documents[0]` den Wert **null**.
Der „Wert“ `null` ist von der Ziffer 0 oder einem leeren String ("") zu unterscheiden.
Er hat die Bedeutung **leer**, **unbestimmt**, „ohne Wert“

! Testen Sie ob ein Dokument geöffnet ist.

Arrays

Arrays sind eine häufig verwendete Datenstruktur in JavaScript. Mit **Arrays** können Listen von Zahlen oder Zeichenketten gesammelt/zusammen verwaltet werden.

- Ein Array ist eine Liste von Einzelelementen
- Definition über Variablendeklaration:
  `var _namen = ["Markus", "Christian", "Thomas"];`
- Oder als leeres Array
  `var _namen = new Array();`
- Anhängen von neuen Elementen
  `_namen.push("Markus");`
- Elemente eines Arrays werden die Position (index) in eckigen Klammern adressiert.
  Das Ergbnis von `_namen[0]` ist `"Markus"` von `_namen[2]` ist `"Thomas"`
- Arrays haben viele nützliche Funktionen/Eigenschaften
  Die Länge eine Arrays (d.h. die Zahl der Elemente): `_namen.length`
  Arrays sortieren: `_namen.sort();`
- Ein Array kann mit `join()` zu einem String zusammengeführt werden.
- Strings können mit `split("TRENNER")` in Array überführt werden

Arrays

- Die Dateien befinden sich im Ordner `01_jsx`
- Öffnen Sie die Datei `anordnungen.jsx` im Extended Script Toolkit
- Verwenden Sie den Objektmodell Viewer zur Übersicht über die Funktionen von Arrays.

! Ermitteln Sie die Länge des Arrays und geben Sie diese aus.

! Fügen Sie ein neues Element hinzu! Prüfen Sie das Ergebnis.

! Geben Sie die Inhalte des Arrays aus.

! Sortieren Sie den Array und geben Sie die Inhalte aus. Testen Sie die Sortierung mit deutschen Umlauten.

! Eine Sonderform ist der **assoziative Array**. Es verwendet keinen numerischen Index, sondern sogenannte Schlüssel zur Indizierung und damit zur Adressierung der Elemente (Wörterbuch). In JavaScript kann anstatt eines Zahlenindex ein String verwendet werden.
Erstellen Sie einen assoziativen Array!

Schleifen

Eine Schleife ist eine Kontrollstruktur, mit der man eine Gruppe von Anweisungen mit einer bestimmten Anzahl von Wiederholungen ausführen kann.

- Sehr oft gebraucht und einfach sind **for-Schleifen**, es gibt noch **while-Schleifen**
- Benötigt wird ein immer **Startwert**, eine **Endbedingung (Abfrage)** und die **Anweisungen** die ausgeführt werden sollen.
  z.B. Alle Seiten eines Dokuments sollen nummeriert werden:
  Startwert 1 (Erste Seite)
  Endbedingung Anzahl der Seiten erreicht
  Anweisung Seiten nummerieren
- Beispiel:

```
for (var i = 0; i < _array.length; i++) {
    //Anweisungsblock
}
```

- **Arrays** (und Sammlungen) eignen sich perfekt für Schleifen
  Arrays haben einen Startwert (Es wird von 0 gezählt)
  Arrays haben eine Endbedinung, die Länge (Eigenschaft `length`) die abgearbeitet werden muss

Schleifen Übung

- Die Dateien befinden sich im Ordner `01_jsx`
- Öffnen Sie die Datei `schleifen.jsx` im Extended Script Toolkit

! Schreiben Sie eine Funktion zum nummerieren von Seiten
Die Sammlung (ähnlich einem Array) von Seiten erhalten Sie mit `app.documents[0].pages`;

! Bauen Sie einen Text aus aktueller Seitenzahl und Gesamtseitenlänge in der Form Seite 1 von 10.

! Legen Sie einen Array mit vier Namen an. Fügen Sie den Array in einem Textrahmen zusammen ohne die Funktion join() zu verwenden.

! Platzieren Sie 5 Textrahmen mit beliebigem Inhalt auf einer Seite!

Funktionen/Methoden

- Methoden von Objekten „tun etwas“ z.B. haben viele Objekte die Methode `.add(...)`
- Funktionen können auch selber geschrieben werden z.B. um Codeteile öfter zu verwenden!
- Machen den Code übersichtlicher!
- Es können **Argumente** (Parameter) übergeben werden, die innerhalb der Funktion verwendet werden können.
- Funktionen können einen **Rückgabewert** der mit der Anweisung `return` zurückgegeben wird `return` beendet immer die Funktion.
- Wir verwenden schon die ganze Zeit die Funktion `main()` obwohl das gar nicht nötig wäre!

Kleinigkeiten

- Unterscheidung von Groß- und Kleinschreibung nicht vergessen
- Am Ende jeder Codezeile/Befehls sollte ein **Semikolon** stehen
- **Kommentare**: Text der auf `//` folgt, wird nicht beachtet, ebenso Text zwischen `/*` und `*/`
- **Reservierte Wörter,** der Sprachkern von JavaScript: Im ESTK sind sie durch eine Farbe hervorgehoben
  Beispielsweise `if`, `else`, `return`, `function`, `null`, `true`, `false` und `var`.

Geltungsbereich/Scope

Werden Variablen außerhalb einer Funktion deklariert, so sind sie global verwendbar. D.h. sie sind in dem gesamten Skript erreichbar. Innerhalb einer Funktion deklarierte Variablen gelten auch nur in dieser und sind von außen nicht erreichbar.

```
var a;
a = 1;
function test() {
  var a;
  a=2;
  alert(a); // ergibt 2
}
test();
alert(a); // ergibt 1
```

Interaktion mit dem Anwender

Es gibt drei sehr einfache Funktionen, um während der Ausführung eines Skripts mit dem Benutzer des Skripts zu kommunizieren:

- Mit der Funktion `alert()` kann der Benutzer auf etwas hingewiesen werden.
- Mit `confirm()` kann der Benutzer etwas gefragt werden
  ```
  var _ergebnis = confirm("Sind Sie sicher?");
  ```
- Mit der Funktion `prompt()` können einfache Abfragen gestaltet werden.
  ```
  var _input = prompt();
  ```

Kommunikation

- Erstellen Sie ein neues Skript.

! Testen Sie die drei Interaktionsfunktionen.

! Lassen Sie den Anwender Ihres Programms entscheiden ob ein Textrahmen mit grüner oder roter Hintergrundfarbe erstellt werden soll.
Die Eigenschaft für die Referenz auf den Textrahmen lautet:
`_tf.fillColor = "Name der Farbe";`
Sie müssen zunächst die beiden Farben rot und grün erstellen und bennen!

! Der Funktion `prompt(...)` können bis zu drei Argumente übergeben werden, finden Sie heraus, wozu diese dienen!

! Lassen Sie den Anwender mit der Funktion `prompt()` den Text für einen Textrahmen bestimmen.