



# 2 VON KNÖPFEN UND FELDERN

## Inhalt

- 2|1. App-Icon für den Launcher erstellen
- 2|2. Verwendung von Buttons und Settern
- 2|3. CRASH-Kurs: JavaScript
- 2|4. Eingabefelder richtig nutzen

Im letzten Kapitel haben wir uns auf die Entwicklung vorbereitet, nun geht es richtig los. Um in diesem Kapitel einen guten Ausgleich zwischen Praxis und Theorie zu schaffen, arbeiten wir einerseits an der „Hello World“-App aus *Kapitel 1* weiter, andererseits beschäftigen wir uns mit Buttons und Settern, Events und der Icon-Erstellung für den Launcher. Ferner erhalten Sie in diesem Kapitel einen JavaScript Crash-Kurs, um Kapitel 2|4 besser verstehen zu können.

### 2|1 App-Icon für den Launcher erstellen

Ein App-Icon sollte die Größe 64 x 64 Pixel besitzen. Für die Icon-Erstellung kann man gut die üblichen Programme, wie Photoshop, GIMP und Co. verwenden.

Das Icon sollte an den Ecken leicht rund sein, um dem Standard von webOS zu entsprechen.

Ist das Icon erstellt, kann man es einfach in Ares hochladen. Zuvor sollte allerdings die Datei `icon.png` entfernt werden.

Ein fertiges Icon könnte so aussehen:



### 2|2 Verwendung von Buttons und Settern

*Buttons* zählen neben den *labels* zu den wichtigsten Elementen bei der Programmierung – und zwar in jeder Sprache. Mit Buttons kann man die Verbindung zwischen dem Programm und dem Anwender herstellen. Der Anwender kann Aktionen starten, beenden, eine Variable ändern u.v.m.

Buttons gehören in der webOS-Programmierung in die Gruppe der Mojo Widgets. Diese Widgets sind dynamische Elemente, die einfach in eine Applikation integriert werden können. Die Besonderheit dabei ist, dass z.B. jeder Button gleich aussieht. Natürlich unterscheiden sich die jeweiligen Buttons durch Größe und andere Eigenschaften, aber die Form ist immer gleich.



Wie man sehen kann, gibt es sehr unterschiedliche Ausführungen von Buttons. Die ersten drei Elemente sind normale Buttons, jedoch mit unterschiedlichen Eigenschaften.

Der Button mit dem kleinen Kreis ist ein `ActivityButton`. Der Kreis dreht sich dabei so lange, wie eine bestimmte Aktion ausgeführt wird, beispielsweise ein Update.

Das vorletzte Element gehört zur Untergruppe der Selektoren (dazu später mehr). Mit diesem sogenannten Radio-Button kann sich der Anwender zwischen 3 Zuständen bzw. Aktionen

entscheiden.

Der Toggle-Button erfüllt im Grunde den gleichen Zweck wie der Radio-Button, jedoch wird er überwiegend dazu verwendet, um Optionen ein- bzw. auszuschalten.

Werfen wir nun mal einen Blick auf den HTML-Code eines Standard-Buttons. Da wir aber mit Ares programmieren, müssen wir diesen Code nicht eingeben, um einen Button zu erstellen. Wir ziehen einfach das Button-Widget in unser Programm in Ares. Ares erstellt dabei den Code.

```
<div x-mojo-element="Button" id="button1"></div>
```

Bei diesem Code wird das Mojo-Widget `Button` mit der internen ID `button1` erstellt. Der Code ist für uns eigentlich nicht wichtig und wird hier nur zur Vollständigkeit halber erklärt. Die interne ID ist aber auch für uns wichtig. Jedes Mal, wenn wir einen Button erstellen, richtet Ares beginnend mit 1 eine ID für diesen Button ein. Diese können wir im rechten Bereich von Ares unter [COMMON](#) → [name](#) jederzeit ändern, jedoch kann der Button dann nicht mehr unter dem Namen angesprochen werden. Ist sonstiger Code mit der alten Button-ID assoziiert, kann der Button nicht mehr angesprochen werden. Deshalb sollte man die ID sofort nach der Erstellung ändern. Je größer die Applikation wird, desto sinnvoller ist es, die Buttons nach logischen Namen zu benennen, z.B. `start_update`.

Erinnern wir uns nun mal an die „Hello World“-App aus Kapitel 1 zurück. Die Anwendung stelle einfach eine Überschrift, einen sogenannten *header*, ein Bild (*picture*) und einen Text (*label*) dar.

Wenden wir unser Wissen doch direkt an und internationalisieren unsere Anwendung. Ziel wird es sein, drei Buttons zu erstellen, die bei einem Tap den Text des `label1` verändern.

Damit die Buttons untereinander auf eine Fläche von 320 x 480 Pixeln passen, verkleinern wir das Bild mit dem Globus ein wenig. Nun ziehen wir einen Button links aus der Widget-Liste in die Mitte unter unser Label. Das wiederholen wir zwei Mal.

Um die Anwendung weiterhin übersichtlich zu halten, können wir aus der Widget-Liste einen *divider* zwischen dem Label und den Buttons einbauen. Im rechten Bereich kann man bei [label1](#) den Namen ändern, beispielsweise in *Language* oder *Sprache*.

Die Buttons beschriften wir auf die gleiche Art mit *Deutsch*, *English* und *français* (das Cedille kann man mit der Tastenkombination [ALT](#) + [135](#) erstellen).

Der erste Entwurf sollte nun in Etwa so aussehen:



Nun, da der graphische Part erledigt ist, geht es an die Programmierung. Drücken wir beispielsweise auf den Button „Deutsch“, soll der Text unter dem Globus in deutscher Sprache erscheinen. Bei den anderen beiden Buttons ebenso.

In Ares öffnen wir dazu im rechten Bereich den Reiter [Events](#) und klicken im Feld [ontap](#) auf das kleine Dokumentensymbol. Die graphische Ansicht des Programms ändert sich in die Code-Ansicht.

Ares hat nun automatisch folgende Codezeile erstellt:

```
button1Tap: function(inSender, event) {  
  
    }  
}
```

Das Programm „achtet“ nun darauf, ob der Button gedrückt wird und führt dann den Code zwischen den beiden geschweiften Klammern aus. Bisher steht dort noch nichts, aber das soll sich nun ändern. Mit dem sogenannten *Setter* kann man Werte von anderen Elementen und Widgets verändern, was man normalerweise über die rechte Seitenleiste tut. Nur erledigt das Programm das selbst im Betrieb.

```
this.$.label1.setLabel("Hallo Welt!");
```

Dieser Setter ändert den Wert **Label** von *label1* in *Hallo Welt*.

Dies wiederholen wir nun für die anderen beiden Buttons. Natürlich muss der Text diesmal mit *Hello World!* bzw. mit *Bonjour tout le monde!* übersetzt werden.

Ist das erledigt, starten wie die Anwendung im Browser-PopUp, im Emulator oder direkt auf dem Gerät. Haben Sie alles richtig gemacht, erscheint nun nach einem Druck auf einen der drei Buttons der Text „Hello World!“ in der jeweiligen Landessprache.

## 2|3 JavaScript Grundlagen

Kommen wir zuerst zu den Elementen, welche man am häufigstem benötigt:

### Konstanten und Variablen

Bei fast allen Anwendungen muss mit Zahlen, Werten und Texten gerechnet werden, die später auch angezeigt werden müssen. Sogenannte *Konstanten* werden im Quellcode definiert und sind anschließend nicht veränderbar. Hier ein Beispiel:

```
Label1.setLabel(„Hallo Welt“);
```

Natürlich lässt sich das Label jederzeit ändern, jedoch steht der Text immer fest.

Anders sieht es bei Variablen aus, denn es gibt auch Anwendungen, bei der Werte nicht vorgegeben sind und erst durch den Anwender definiert werden. Nehmen wir als Beispiel ein Programm, welches den Anwender nach seinem Alter fragt und dieses anschließend wieder ausgibt. Zuerst benötigen wir also den Datentyp für unser *alter*. In der Namensgebung sind wir überwiegend frei.

```
var alter;
```

Wir könnten unserem *alter* auch direkt einen Wert zuweisen:

```
var alter = 21;
```

Die Definition *var* ist nur bei der *Deklaration* nötig. Einmal durchgeführt, können wir den Wert von *alter* so ändern:

**Tabelle 1 | Die wichtigsten Datentypen im Überblick**

Boolean	Wahrheitswert (true & false)
Number	Zahlenwert (8 Byte lange Fließkommazahl )
String	Zeichenkette

```
alter = 12;
Label1.setLabel(„Du bist „ + alter + „Jahre alt“);
```

Der Datentyp *var* steht für eine Variable.

Unter JavaScript reicht dieser Datentyp auch aus, denn er erkennt je nach zugewiesenem Inhalt automatisch um welchen Datentyp es sich handelt.

## Datentypen und Ihre Anwendung

### Rechenoperatoren:

Einmal definiert kann man mit Variablen rechnen wie man will.

Hier ein paar wichtige Rechenoperatoren:

```
var zahl1 = 10;
var zahl2 = 12;
var ergebnis;
```

**Tabelle 2 | Rechenoperationen**

Addition	ergebnis = zahl1 + zahl2;
Subtraktion	ergebnis = zahl1 – zahl2;
Multiplikation	ergebnis = zahl1 * zahl2;
Division	ergebnis = zahl1 / zahl2;
Um Eins erhöhen (verringern)	zahl1 ++; (--)

Es lässt sich auch der Restwert bei der Division zweier Zahlen ermitteln:

```
ergebnis = zahl1 % zahl2;
```

Der Operator % nennt sich Modulo.

## If und else

Bei jedem Programm müssen einige Werte überprüft werden. Nehmen wir als Beispiel das vorhin besprochene Alter-Programm. Sollte der Nutzer beispielsweise jünger als 18 Jahre sein, dann soll er möglicherweise einen anderen Text zu sehen bekommen:

```
if(alter > 18)
{labell.setText(„Du bist alt!“);}
else if (alter < 18)
{labell.setText(„Du bist jung!“);}
else if (alter == 18){ labell.setText(„Du bist 18!“); }
```

**Tabelle 3 | Vergleichsoperatoren**

==	gleich
<=	kleiner gleich
>=	größer gleich
!=	ungleich

## Schleifen

Oft müssen bestimmte Rechenanwendungen mehrmals hintereinander durchgeführt werden.

Als Beispiel nehmen wir ein Programm, welches uns eine Vielzahl von Quadratzahlen anzeigt:

```
zahl = 2;
```

```
label1.setText(zahl*zahl);  
zahl++;  
label2.setText(zahl*zahl);
```

Auf Dauer würde unser Programm zu einer unmöglichen Schreibarbeit werden.

Zum Glück gibt es Schleifen, welche uns hier weiterhelfen:

```
var anzeige;  
for(var i = 1; i <= 10; i++)  
{  
    zahl = i*i;  
    anzeige = (anzeige + zahl + ",");  
}  
  
label1.SetLabel(anzeige);
```

Unser Programm würde nun folgendes Anzeigen:

**1, 4, 9, 16, 25...**

Das Prinzip der for-Schleife ist einfach:

In den Klammern steht die Bedingung. Und zwar legt sie zuerst eine Variable *i* fest, welche am Anfang den Wert *1* zugeteilt bekommt. Als nächstes folgt die Bedingung für die Ausführung der Schleife, nämlich dass *i* kleiner oder gleich *10* sein soll. Die letzte Anweisung sorgt dafür, dass *i* am Ende des Durchlaufs um *1* erhöht wird. Dadurch wird unsere Schleife insgesamt *10*-mal durchlaufen.

## Funktionen

Funktionen dienen dazu, bestimmte sich wiederholende Rechenschritte auszulagern. Eine Funktion wird außerhalb des Programmbereichs (*Main*) festgelegt.

Beispiel einer Funktion:

```
function addieren()  
{  
  label1.setLabel(2+3);  
}
```

Aufruf im Hauptprogramm:

```
addieren();
```

Dies ist auch die Grundfunktionsweise von Ares-Events. Die Funktion *Button\_Click* wird automatisch definiert und der Benutzer kann bestimmen, welche Aktionen ausgeführt werden sollen.

Da man oft mit Werten aus dem Hauptprogramm rechnen muss, ist es wichtig, dass man einige Werte den Funktionen übergibt.

Dies funktioniert folgendermaßen:

### Hauptprogramm:

```
zahl1 = 5;  
zahl2 = 3;  
  
addieren(zahl1, zahl2);
```

### Funktion:

```
function addieren(var zahl1, var zahl2)  
{  
  Label1.setLabel(zahl1 + zahl2);  
}
```

Damit wäre genug zu JavaScript erläutert. In den folgenden Kapiteln werden wir automatisch durch Arbeiten an Projekten weitere Beispiele, Zusätze und Ausführungen kennen lernen.

## Aufgaben

### 2|4 Eingabefelder richtig nutzen

Mit diesem Kapitel möchten wir einen neuen Bereich vorstellen: in jedem Kapitel stellen wir Ihnen ein paar Aufgaben. Die Lösungen dazu werden dann auf [dev.tamspalm.de](https://dev.tamspalm.de) veröffentlicht.

#### A1 | Geben Sie dem Anwender die Möglichkeit, den *HelloWorld*-Text selbst zu ändern.

Erstellen Sie oberhalb der Sprachen-Buttons ein *Group*-Element und benennen Sie dieses mit einem aussagekräftigem Namen. Ziehen Sie in dieses Element ein *Row*-Element. In diesem Element erstellen Sie ein *TextField*. Bitte teilen Sie dem Nutzer noch mit, was er denn überhaupt eingeben soll (Tipp: Nutzen Sie hierfür die *hint*-Funktion).

Entscheiden Sie sich nun für ein Ihrer Meinung nach geeignetes Event und übergeben Sie den Inhalt des *TextFields* an das *HelloWorld*-Label.

#### Tipps |

1. Nutzen Sie folgenden Code, um den Inhalt eines Textfeldes an eine Variable zu übergeben:

```
var customtext = this.$.textField1.getValue();
```

2. Nutzen Sie den in 2|2. vorgestellten Setter, um den Inhalt der Variablen *customtext* an das *HelloWorld*-Label weiterzugeben.