7 Plugins einbinden

PhoneGap bringt einen standardisierten Hardwarezugriff für Smartphones mit. Aber was passiert, wenn Sie mehr wollen: Wenn Sie eine Hardware per Bluetooth ansprechen wollen oder Features der jeweiligen Plattform nutzen möchten? Denken Sie an Beispiele wie das Scannen von Barcodes oder die native Twitter-Integration in Apples iOS. Für solche Anforderungen sind Plugins vorgesehen.

Plugins kapseln Hardware oder native Softwarezugriffe in JavaScript und machen diese unter PhoneGap zugänglich. Das Entwickeln eines Plugins setzt daher immer Plattformcode voraus – für das iOS etwa Objective-C. Eine zusätzliche JavaScript-Datei bindet dabei die Aufrufe an die nativen Funktionen. Damit Sie nicht alles selber entwickeln müssen, stellt das GitHub-Verzeichnis[1] ein zentrales Repository für Plugins dar. Allerdings findet sich auch so manches Plugin über eine Google-Suche.

Ich habe für Sie drei Plugins ausgewählt, die wir uns näher anschauen wollen. Dabei lernen Sie, wie Plugins für die Plattformen Android und iOS einzubinden und aufzurufen sind. Es handelt sich zum Anfang um die beliebte Taschenlampen-App, also den Zugriff auf die LED des iPhones. Danach wird es seriöser mit dem Einsatz eines Barcodescanners. Abschließen möchte ich mit einem Projekt, das die Möglichkeiten von Plugins sehr gut beschreibt: einem Augmented-Reality-Viewer, der mit nur drei Zeilen JavaScript-Code einzubinden ist und damit zeigt, wie einfach Komplexität aus Anwendungen genommen werden kann.

Zusammengefasst – mit Plugins macht die Anwendungserstellung mit PhoneGap noch mehr Spaß. Diese bieten die oft vermissten Features der nativen Plattformen. Aber Achtung: Damit entfernen Sie sich auch vom Prinzip »Ein Sourcecode = viele Plattformen«.

7.1 Beispiel »Die Taschenlampe«

Seit der vierten Gerätegeneration ist Apples iPhone mit einer LED als Blitz ausgestattet, die man auch prima als Taschenlampe nutzen kann. Diese Funktion werden wir nun mithilfe des »Torch«-Plugins für PhoneGap nachstellen. Ganz ne-

1. *https://github.com/phonegap/phonegap-plugins.git*

benbei lernen wir die grundlegende Nutzung eines Plugins kennen. Wichtig ist dabei zu wissen, dass es sich hier ausschließlich um ein iOS-Feature handelt. Diese Funktion ist nicht unter anderen Plattformen mit diesem Plugin möglich. Für den Einstieg ist es ein gutes kompaktes Beispiel.

7.1.1 Die Zutaten

Das PhoneGap-Torch-Plugin ist in der offiziellen Plugin-Sammlung[2] erhältlich. Da es sich hier um ein GitHub-Repository handelt, können Sie es wahlweise klonen oder als Zip-Datei herunterladen. Natürlich darf PhoneGap auch nicht fehlen. Für den weiteren Verlauf entpacken Sie bitte die Zip-Datei. Für das folgende Rezept benötigen wir Dateien aus dem Plugin-Ordner `Torch`. Diesen finden Sie unterhalb des Ordners `iPhone` in der Zip-Datei bzw. im geklonten Repository.

Abb. 7–1 *Torch-Plugin-Ordner*

Alternativ können Sie auch die aktuelle Fassung im GitHub-Repository dieses Buches[3] finden. Jetzt können wir mit der Taschenlampen-App starten.

7.1.2 Das Rezept

Erstellen Sie als Erstes ein neues PhoneGap-Projekt. Als Namen vergeben wir hier *meineTaschenlampe*. Daher würden wir das Projekt mit folgendem Aufruf auf der Kommandozeile anlegen:

```
create meineTaschenlampe de.phonegapbuch.meineTaschenlampe meineTaschenlampe
```

Jetzt gibt es ein neues XCode-Projekt. Starten Sie also XCode und öffnen die Projektdatei.

Da wir ein Plugin nutzen möchten, sind noch Dateien in das Projekt hinzuzufügen. Zunächst kopieren Sie bitte die Datei `Torch.js` in den `www`-Ordner. Hiermit haben Sie sozusagen die API-Bibliothek für das Plugin bereitgestellt. Wir könnten nun bereits loslegen, die vorhandene Datei `index.html` mit Plugin-Aufrufen in JavaScript anzupassen. Allerdings fehlt noch die notwendige Erweiterung auf der

2. *https://github.com/phonegap/phonegap-plugins.git*
3. *https://github.com/phonegapbuch/plugins.git*

Objective-C-Seite. Dafür nutzen Sie die Dateien `Torch.h` und `Torch.m` aus der Zip-Datei und kopieren diese in den `Plugins`-Ordner Ihres Taschenlampenprojekts.

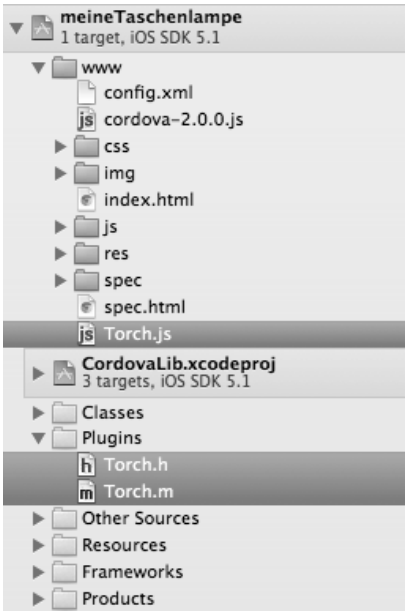

Abb. 7–2 *XCode-Projektübersicht beim Torch-Beispiel*

Bevor es jedoch losgeht, fehlt noch eine Kleinigkeit, denn PhoneGap weiß noch nichts von dem Plugin, in unserem Fall von den Implementierungen auf der Objective-C-Seite. Damit wir gleich keine Fehlermeldung bekommen, muss noch die Datei `Cordova.plist` erweitert werden. Sie finden diese Datei im `Resources`-Ordner des XCode-Projekts.

Im Bereich *Plugins* erstellen Sie einen neuen Schlüssel mit dem Namen *Torch*. Als Wert wird hier auch *Torch* hinterlegt. Damit weiß PhoneGap nun auch von dem Objective-C-Teil und kann ihn beim Kompilieren der App verwenden. Jetzt kann es endlich mit dem Code losgehen.

Um das Plugin schnell auszuprobieren, können Sie das Listing 7–1 nutzen. Kopieren Sie dabei das gesamte Listing in die Datei `index.html`. Beachten Sie bitte, dass ein Test im Simulator natürlich ohne Erfolg sein wird, da nur ein echtes iPhone 4 oder 5 eine LED besitzt.

▼Plugins	Dictionary	(17 items)
Torch	String	Torch
Device	String	CDVDevice
Logger	String	CDVLogger
Compass	String	CDVLocation
Accelerometer	String	CDVAccelerometer
Camera	String	CDVCamera
NetworkStatus	String	CDVConnection
Contacts	String	CDVContacts
Debug Console	String	CDVDebugConsole
File	String	CDVFile
FileTransfer	String	CDVFileTransfer
Geolocation	String	CDVLocation
Notification	String	CDVNotification
Media	String	CDVSound
Capture	String	CDVCapture
SplashScreen	String	CDVSplashScreen
Battery	String	CDVBattery

Abb. 7–3 *Cordova.plist-Ansicht in XCode*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
      maximum-scale=1.0, user-scalable=no;" />

    <meta http-equiv="Content-type" content="text/html; charset=utf-8">
    <script type="text/javascript" charset="utf-8"
      src="cordova-2.0.0.js"></script>
    <script type="text/javascript" charset="utf-8" src="Torch.js"></script>
    <script type="text/javascript" charset="utf-8">
      function onBodyLoad()
      {
         document.addEventListener("deviceready", onDeviceReady, false);
      };
      function onDeviceReady()
      {
         setInterval(function() {
            document.getElementById("on").innerHTML =
              window.plugins.torch.isOn? "AN" : "AUS";
         }, 500);
      };
    </script>
  </head>
  <body onload="onBodyLoad()">
      <h1 style="font-family:Helvetica Neue">Taschenlampe</h1>
      <div>Die Lampe ist: <span id="on"></span></div>
```

```
      <button style="background-color:lightgreen;width:100%;height:40px;
        margin-top:50px;" onclick="window.plugins.torch.turnOn()">An</button>
      <button style="background-color:red;width:100%;height:40px;
        margin-top:100px;" onclick="window.plugins.torch.turnOff()">Aus</button>
  </body>
</html>
```

Listing 7–1 *Torch-Plugin-Beispiel*

Jetzt ist alles vorbereitet und wartet auf Ausführung. Starten Sie das Erstellen und Deployen auf Ihr angeschlossenes iPhone. Nach kurzer Zeit sollten Sie Folgendes sehen:

Abb. 7–4 *Taschenlampen-Menü*

Nun können Sie mit den beiden Flächen *An* und *Aus* die LED schalten. Glückwunsch, Ihr erstes Plugin ist erfolgreich eingebunden.

Ich möchte noch kurz auf den Code eingehen, der das Plugin ansteuert. Die Datei `index.html` hat wie in anderen Projekten auch erst einmal wieder die `cordova-2.0.0.js`-Datei geladen. Danach folgt dann die Datei `Torch.js`, die für die Funktionen des Plugins zuständig ist. Das obligatorische Event `deviceready` wird abgewartet, um dann ein Intervall von 500 ms festzulegen. Dieses Intervall beeinflusst den Wert des HTML-Containers, der mit der ID `on` versehen ist. Hier ist dann auch schon der erste Aufruf des Plugins zu sehen: `window.plugin.torch.isOn`. Dieser Aufruf im Intervall bewirkt, dass alle 500 ms abgefragt wird, ob die LED an ist. Je nach Status weist dann der ternäre Operator `?` den Wert `AN` oder `AUS` zu. Dieser Wert wird im HTML-Container mit der ID `on` angezeigt. Zum eigentlichen

Ein- bzw. Ausschalten der LED wird auf den beiden Buttons im HTML-Body ein `onClick` mit der Plugin-Funktion `window.plugins.torch.turnOn()` bzw. `turnOff()` hinterlegt.

So einfach ist die Nutzung eines Plugins. Sicherlich ist eine Taschenlampe keine große Sache, aber das Beispiel zeigt sehr gut, wie einfach Plugins genutzt werden können. In den nächsten beiden Abschnitten werden Sie ernsthaftere Beispiele kennenlernen.

7.2 Beispiel »Barcode-Reader«

Nach dem Freizeitbeispiel der Taschenlampe möchte ich nun mit Ihnen ein Businessbeispiel für ein Plugin durchgehen: eine Barcode-Leser-Anwendung. Barcodes finden sich überall. Strichcodes machen es uns einfach, lange Zahlen oder auch kleine Datenmengen schnell zu erfassen. Ein Beispiel gefällig? Sie können eine Anwendung zur DVD-Archivierung mit einem Barcodescanner zur einfachen Erfassung erweitern. Damit braucht der Anwender nicht erst den Titel jeder DVD einzutippen – der EAN13-Strichcode und ein Webservice reichen aus.

Abb. 7–5 *EAN13-Strichcode*

Ein einfaches Scannen übernimmt das lästige Eintippen der Daten. Auch die Verwendung von QR-Codes nimmt zu. Diese Punkt-Quadrate können nicht nur eine Zahlenfolge darstellen, sie speichern auch noch mehr Daten.

Abb. 7–6 *Beispiel QR-Code*

Zum Beispiel können damit ganze Hyperlinks oder Visitenkarten abgebildet werden. Daher ist das Scannen und Verarbeiten von Strichcodes sicherlich für viele Anwendungen interessant. PhoneGap hat zwar keinen Barcodescanner in der API integriert, aber mittels eines Plugins können wir diese Funktionalität erreichen. Die verwendete Scan-Engine unterstützt dabei folgende Formate: