3 Die Zielplattformen

In diesem Kapitel werden Sie das Aufsetzen der Entwicklungsumgebungen der verschiedenen Smartphone-Systeme kennenlernen. Jede Smartphone-Plattform stelle ich Ihnen dabei in einem eigenen Unterkapitel vor. Hier wird zuerst die Installation der Entwicklungsumgebung und Tools gezeigt. Danach wird ein kleines Beispiel für die jeweilige Plattform zum Laufen gebracht. Außerdem lernen Sie auch mögliche Fallstricke kennen und erfahren nützliche Tipps im Umgang mit den Tools. Am Ende dieses Kapitels sind Sie in der Lage, für jedes von PhoneGap unterstützte System eine Entwicklungsumgebung einzurichten und lauffähige Projekte zu erstellen, die Ihre App in das jeweilige Smartphone-Format verpackt. Mit diesen erzeugten Binärpaketen können Sie Ihre App in den jeweiligen Stores, Marketplaces oder App-Shops anbieten. Sie werden sehen, dass sich der Einrichtungsaufwand dabei je nach Zielplattform stark unterscheidet.

Denken Sie daran, Sie können die Lesereihenfolge in diesem Kapitel frei wählen. Interessieren Sie sich nur für iOS oder Android als Plattform? Kein Problem, jeder Abschnitt ist für sich alleine lesbar und endet mit einem eigenen Testprojekt.

Nur der folgende Abschnitt 3.1 ist verpflichtend, denn wir brauchen eine Test-App, die wir zu Beginn dieses Kapitels erstellen. Dabei lernen Sie auch schon die ersten PhoneGap-Befehle kennen.

3.1 »myDevice« – Ihre erste Cross-Plattform-App

Um uns mit PhoneGap praktisch zu beschäftigen, benötigen wir zunächst eine kleine App als Beispielanwendung. Diese Anwendung wird dann für alle Plattformen genutzt. Also haben wir hier das Prinzip: ein Quellcode (Source) = viele Plattformen. Die Beispielanwendung soll dabei die Gerätedaten auslesen und anzeigen. Sicherlich keine komplizierte Anwendung, aber Sie werden dabei bereits die ersten JavaScript-Aufrufe von PhoneGap nutzen und deren Arbeitsweise kennenlernen. Als Grundlage ist hier ein übersichtliches Skript vorbereitet.

```
<!DOCTYPE html>
<html>

  <head>
    <title>Gerätedaten-Beispiel</title>
    <script type="text/javascript" charset="utf-8"
    src="cordova-2.0.0.js"></script>
    <script type="text/javascript" charset="utf-8">
      // Warten auf PhoneGap
      document.addEventListener("devicereadyready", onDeviceReady, false);

      // PhoneGap ist geladen.
      function onDeviceReady() {
        var element = document.getElementById('deviceProperties');

        element.innerHTML = 'Device Name: ' + device.name + '<br />' +
          'Device PhoneGap: ' + device.cordova + '<br />' +
          'Device Platform: ' + device.platform + '<br />' +
          'Device UUID: ' + device.uuid + '<br />' +
          'Device Version: ' + device.version + '<br />';
      }
    </script>
  </head>

  <body>
    <p id="deviceProperties">Laden der Gerätedaten...</p>
  </body>

</html>
```

Listing 3–1 *Gerätedaten per PhoneGap ermitteln*

Lassen Sie uns den Code kurz Schritt für Schritt durchgehen.

Erst einmal laden wir die PhoneGap-eigene JavaScript-Bibliothek. Sie trägt seit der Übergabe an die Apache Foundation den Namen `cordova-x.x.x.js`.

```
<script type="text/javascript" charset="utf-8" src="cordova-2.0.0.js"></script>
```

Gleich nach dem Laden wird ein HTML-konformer Eventlistener registriert.

```
document.addEventListener("devicereadyready", onDeviceReady, false);
```

Dabei wird auf das Ereignis `devicereadyready` reagiert und dann die Funktion `onDeviceReady` aufgerufen. Diese Zeile werden Sie in jeder Ihrer Anwendungen nutzen, da Sie hierduch sicher sein können, dass PhoneGap komplett geladen ist. Dieses Ereignis (Event) wird von PhoneGap selbst erzeugt. Es ist ja sowohl eine native Codebasis als auch eine JavaScript-Bibliothek. Während der native Codeteil initialisiert wird, kann es sein, dass der JavaScript-Teil bereits komplett geladen und ausgeführt ist. Dabei könnte es passieren, dass Sie per JavaScript bereits auf Funktionen zugreifen, deren nativer Codeteil noch nicht bereit ist. Ihre Aufrufe

werden ja von JavaScript in plattformspezifische Aufrufe umgewandelt und an die Plattform gesendet. Um diesen möglichen Timingproblemen zuvorzukommen, wird das Event `deviceready` von PhoneGap erst dann an den Browser geschickt, wenn auch das native Backend vollständig geladen ist. Daher ist das Warten auf dieses Event eine Vorsichtsmaßnahme.

Aber nun wieder zur eigentlichen Funktion `onDeviceReady`. Als Nächstes folgt ein Bezug auf ein HTML-Element.

```
var element = document.getElementById('deviceProperties');
```

Das Element mit der ID `deviceProperties` im HTML-Dokument wird dann die ermittelten Gerätedaten anzeigen. Der Zugriff auf diese Daten erfolgt eine Zeile danach. Nehmen wir dort beispielsweise die Zeile:

```
'Device Platform: ' + device.platform
```

Hier sehen Sie den Aufruf von `device.platform`. Dieser Aufruf ist aus der PhoneGap-API. Je nach Plattform erhalten Sie dann die Werte `Android`, `BlackBerry`, `iPhone` oder `webOS`. Auch wenn Apple mittlerweile iOS als Plattform angibt, wird an dieser Stelle immer noch `iPhone` gemeldet. Dies ist allerdings ein Fehler, der vom Apple SKD stammt und nicht von PhoneGap.

Diese kleine Beispielanwendung soll uns nun für das Setup der jeweiligen Plattformen als Test begleiten. Am Schluss der folgenden Abschnitte soll dieses Beispiel auf den jeweiligen Systemen laufen. Dabei wird der gezeigte Code nicht geändert. Es ist also ein Quelltext für fünf Smartphone-Systeme.

Das deviceready-Event

Es ist obligatorisch, auf das Ereignis `deviceready` zu warten. Erst danach ist es sicher, PhoneGap-API-Aufrufe auszuführen, da erst dann auch der native Codeteil geladen ist.

3.2 iOS

Eine der beliebtesten Plattformen für PhoneGap ist leider auch eine, die den meisten Veränderungen unterliegt. In der aktuellen Version 2.0 wurde mit dem Template-Ansatz von XCode gebrochen. Hier hatten Sie eine Projektvorlage in XCode, die Sie konfigurieren mussten. Mit dem Versionssprung auf 2.x wurde diese entfernt und dafür wurden Kommandozeilentools eingeführt. Damit erzeugen Sie nun ein lauffähiges Projekt, das Sie in Apples XCode öffnen und kompilieren können. Der Kommandozeilen-Ansatz ist für die Arbeit mit iOS obligatorisch, kann aber optional auch für die Plattformen Android und BlackBerry eingesetzt werden.

3.2.1 Die Entwicklungsumgebung installieren

Als Grundlage für das Erstellen von iOS-Apps wird Apples eigene Entwicklungsumgebung XCode benötigt. Nur damit ist das Entwickeln bzw. Kompilieren von Apps für iPhone und iPad möglich. Dieses Programm können Sie allerdings ganz einfach in Apples eigenem App Store beziehen.

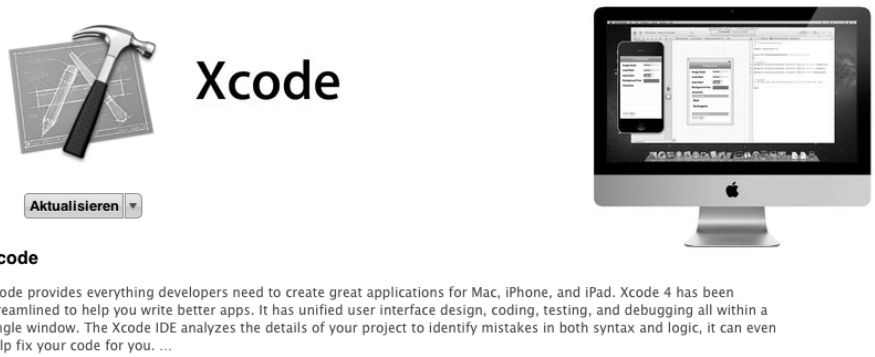

Abb. 3–1 *Apples XCode im App Store*

Damit zeigt sich, dass für die Entwicklung einer nativen Anwendung zwingend ein Mac-OS-Betriebssystem vorausgesetzt wird. Nach dem Download wird XCode automatisch installiert und Sie können mit dem Entwickeln beginnen. Planen Sie für den Download Zeit ein, da er mehrere Gigabyte umfasst.

Zusätzlich sollten Sie noch die Kommandozeilenprogramme von XCode nachinstallieren. Viele Tools, unter anderem auch PhoneGap, nutzen diese im Hintergrund. Die Installation erfolgt dabei in XCode. Dazu gehen Sie bitte per Menüfolge die Schritte *Preferences -> Downloads -> Components -> Command Line Tools* und installieren diese dann.

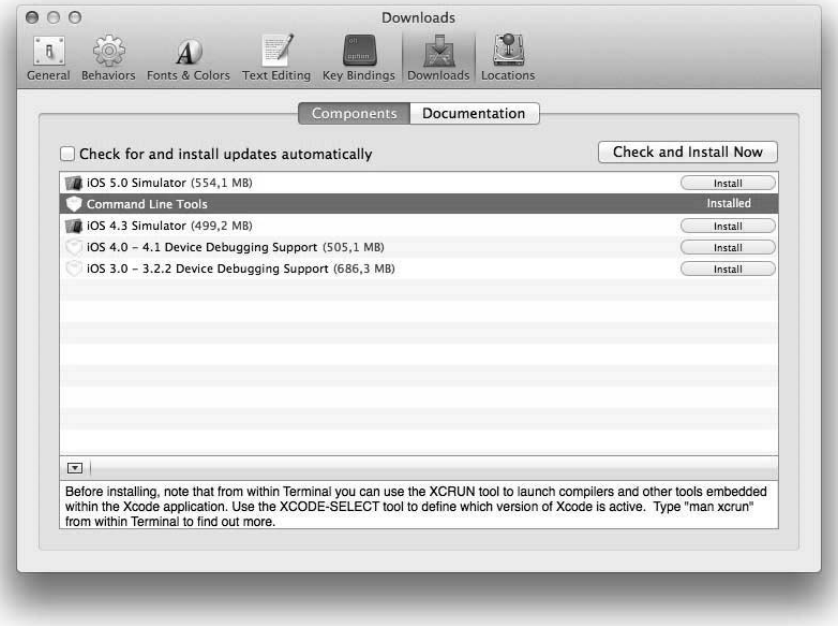

Abb. 3–2 *Kommandozeilentools von XCode*

Sie erhalten dieses unter www.phonegap.com/download. Nach dem Entpacken der Zip-Datei sollte die Ordnerstruktur wie folgt aussehen:

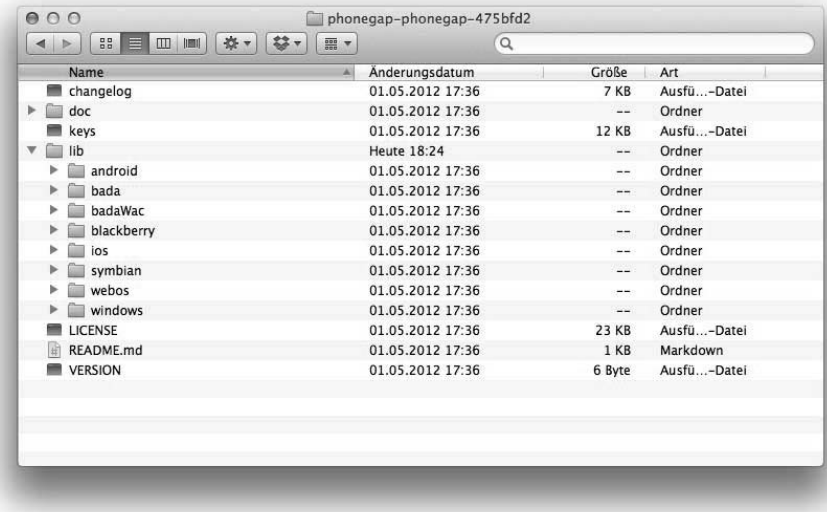

Abb. 3–3 *Ordnerstruktur des PhoneGap-Pakets*

Für die jeweiligen Zielsysteme gibt es ein separates Verzeichnis. Für uns ist in diesem Abschnitt das `ios`-Verzeichnis wichtig. Seit der Version 2.0 von PhoneGap er-

stellen Sie neue Projekte per Kommandozeile. Diese können danach in XCode geladen und bearbeitet werden. Lassen Sie uns dafür die notwendigen Schritte durchgehen.

Im `ios`-Verzeichnis finden Sie das Unterverzeichnis `bin`. Dieses Verzeichnis enthält die notwendigen Kommandozeilentools zum Erstellen eines PhoneGap-Projektes. Kopieren Sie sich daher bitte diesen Ordner auf die Festplatte Ihres Rechners. In unserem Beispiel habe ich ein Verzeichnis `Phonegap` unterhalb meines Home-Ordners angelegt. Generell ist es Ihnen freigestellt, das Ziel zu bestimmen. Allerdings nutze ich in diesem Beispiel den eben genannten Pfad.

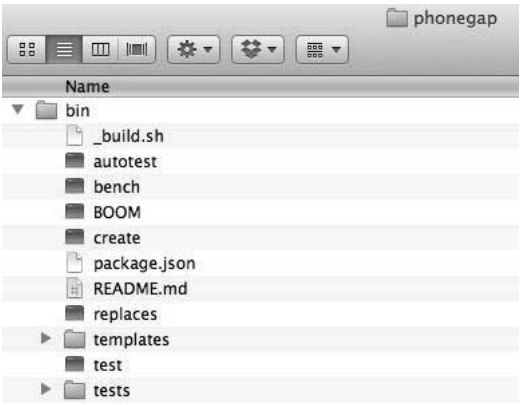

Abb. 3–4 *CLI-Tools von PhoneGap*

Wenn Sie nun die Programme in Ihrem `phonegap/bin`-Ordner aufrufen wollen, müssen Sie immer den kompletten Pfad angeben. Um das zu vereinfachen, fügen wir das `bin`-Verzeichnis der `Path`-Variablen hinzu. Nutzen Sie dafür Ihren Lieblingstexteditor und fügen der Datei `.bash_profile` die Zeile:

```
export PATH=$PATH:~/phonegap/bin
```

hinzu. Speichern Sie dann die Datei. Ein anschließendes `source .bash_profile` bzw. `source .bashrc` lädt den Inhalt der Dateien. Dies erspart Ihnen einen Neustart des Systems bzw. der Terminalsitzung. Damit stehen Ihnen nun die Tools in jedem Pfad zur Verfügung. Bevor wir jedoch eine neue App erstellen können, muss noch die statische PhoneGap-Bibliothek installiert werden. Dafür starten Sie die Datei `Cordova-2.0.0.dmg` aus dem Verzeichnis *ios*.

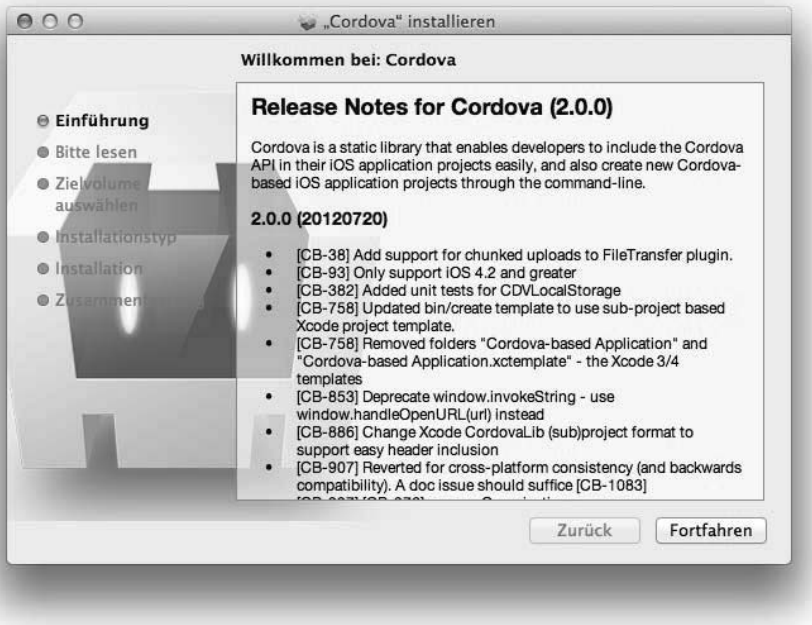

Abb. 3–5 *Setup-Assistent der statischen PhoneGap-Bibliothek*

Folgen Sie einfach dem Assistenten für die Installation. Nach diesem Setup ist nun alles vorbereitet und wir können mit der ersten Anwendung für iOS beginnen.

3.2.2 Die erste App für iOS

Um ein neues Projekt anzulegen, ist die Kommandozeile zu nutzen. Starten Sie daher bitte eine Konsole. Da die PhoneGap-Tools ja in Ihrer Pfad-Variablen hinterlegt sind, können wir direkt mit dem Aufruf starten.

```
create ersteApp de.phonegapbuch.ersteApp ersteApp
```

Damit erstellen Sie ein neues iOS-XCode-Projekt. Dabei ist der erste Parameter der Pfad zum Projekt; in unserem Beispiel ist es also der Ordner *ersteApp*.

Danach folgt der sogenannte Packagename. Er wird dazu genutzt, eine App technisch mit einem einzigartigen Namen zu identifizieren. Sollten Sie bisher noch nicht mit diesem Ausdruck in Kontakt gekommen sein, hier eine kurze Konvention: Normalerweise wird der Packagename mit der *Reverse-Domain-Name*-Notation beschrieben; in unserem Beispiel ist es also

```
de.phonegapbuch.ersteApp.
```

Als letzten Parameter geben Sie noch den Projektnamen für XCode an. Dieser ist in unserem Beispiel auch wieder `ersteApp`. Nach der Erstellung sollten Sie folgenden Inhalt in Ihrem neuen Verzeichnis *ersteApp* vorfinden.

Name	Änderungsdatum	Größe	Art
ersteApp.xcodeproj	Heute 14:59	217 KB	Xcode Project
cordova	Heute 14:59	--	Ordner
ersteApp	Heute 14:59	--	Ordner
www	Heute 14:59	--	Ordner

Abb. 3–6 *Verzeichnisinhalt der iOS-Projektanlage*

Damit haben Sie bereits jetzt eine lauffähige Anwendung erstellt. Lassen Sie uns nun in XCode wechseln und sehen, wie die neue iOS-Anwendung zu kompilieren ist. Ein Doppelklick auf die Datei `ersteApp.xcodeproj` startet Ihr XCode und öffnet das Projekt.

Abb. 3–7 *Das erste Projekt in XCode*

Starten wir nun das erstellte Projekt mit dem *Run*-Befehl. Achten Sie darauf, dass sowohl Ihre Anwendung als auch der Simulator unter dem Abschnitt *Scheme* als Kompilierungsziel ausgewählt ist.

Abb. 3–8 *Auswahl von XCode Run*

Nach einem kurzen Moment sollten Sie den iPhone-Simulator sehen.

Abb. 3–9 *iPhone-Simulator*

Wunderbar: Das erstellte Projekt ist lauffähig. Nun können wir den Beispielcode mit unserer Anwendung ersetzen. Kehren Sie dafür in XCode zurück und lassen Sie uns die Projektstruktur näher betrachten.

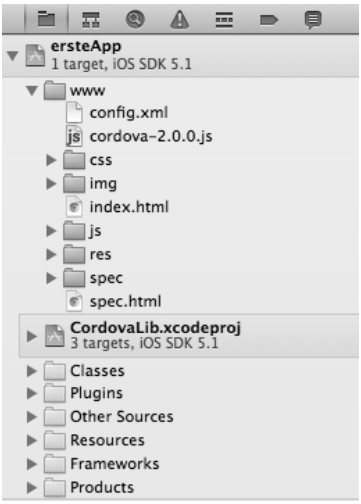

Abb. 3–10 *PhoneGap-Projektstruktur*

Der zentrale Punkt Ihrer Anwendung befindet sich im `www`-Ordner. Hier ist Ihr Anwendungsverzeichnis. Eine vorhandene Datei `index.html` in diesem Ordner wird immer beim Start der App aufgerufen. Auch sollte hier die `cordova-2.0.0.js`-JavaScript-Datei liegen. Diese Bibliothek bietet uns Zugriff auf die Hardware des Smartphones mittels JavaScript. Mehr zu den Zugriffsmöglichkeiten finden Sie in den Kapiteln 4 und 5.

PhoneGap hat bereits, wie Sie gemerkt haben, bei der Projektanlage ein lauffähiges Beispiel nebst Ordnerstruktur für Sie erstellt. Sie können diese für Ihre Projekte gerne übernehmen. Für unsere Anwendung benötigen wir nur die Datei `index.html`. Daher öffnen Sie die Datei bitte in XCode und löschen den gesamten Inhalt. Wir werden diesen mit unserem Anwendungscode aus dem Abschnitt 3.1 ersetzen. Dabei reicht ein einfaches Copy&Paste aus. Jetzt starten Sie erneut per *Run* das Kompilieren und Ausführen der App auf dem Simulator. Nach kurzer Zeit sollten Sie folgendes Ergebnis sehen:

Abb. 3–11 *myDevice-Beispiel auf dem iPhone*

Herzlichen Glückwunsch! Damit läuft nun Ihre eigene Anwendung unter iOS. Sie haben alle notwendigen Schritte für das Erstellen einer neuen App durchgeführt. Die weiteren Kapitel geben Ihnen noch wissenswerte Informationen rund um iOS-Projekte mit PhoneGap. Sie können aber auch direkt in den Abschnitt 3.3 springen und sich die Projekterstellung für Android anschauen.

3.2.3 Der Netzwerkzugriff in iOS

Häufig möchten Sie in Ihrer Anwendung eine Kommunikation mit Servern aufnehmen. Dabei werden Sie merken, dass dies erst einmal nicht funktioniert. Das liegt an der sogenannten `External Hosts`-Tabelle. Aus Sicherheitsgründen kann keine mit PhoneGap erstellte App auf das Internet zugreifen. Da Sie aber sicherlich oft Zugriff benötigen, z.B. für das Laden von Daten eines Webservice, möchte ich Ihnen zeigen, wie Sie die Anwendung entsprechend konfigurieren. Dazu wechseln Sie innerhalb von XCode in der Projektansicht.

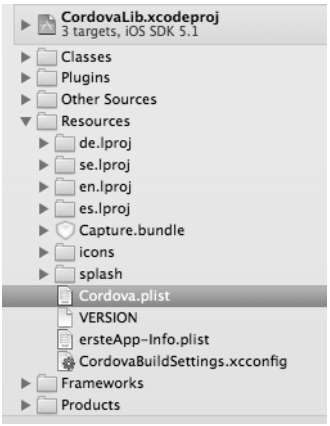

Abb. 3–12 *Projektansicht von XCode*

Hier klappen Sie als Nächstes den Ordner `Resources` auf. Sie finden dort die Datei `Cordova.plist`. Öffnen Sie sie per Klick.

Abb. 3–13 *External Hosts-Abschnitt in der Datei Cordova.plist*

Jetzt können Sie eine Reihe von Einstellungsmöglichkeiten vornehmen. Hier möchte ich Sie auf den Abschnitt *External Hosts* aufmerksam machen. Wenn Sie hier mithilfe des +-Zeichens eine neue Zeile einfügen, können Sie die Serveradresse freigeben. Im obigen Beispiel sind alle Verbindungen zu www.cordovabuch.de erlaubt worden. Mit dieser Modifikation können Sie nun beliebige Adressen freigeben.

3.2.4 Bilder und Icons anpassen

Nachdem die App unter iOS läuft und nun auch mit dem Internet reden kann, fehlt noch etwas. Das Bild im Startbildschirm und auch die vorgegebenen Menü-

Icons wollen Sie sicherlich nicht in Ihr Projekt übernehmen. Hier möchte ich deshalb kurz zeigen, wo Sie diese ändern können. Gehen Sie daher in den Ordner `icons` bzw. `splash` und ersetzen die Dateien durch Ihre eigenen Bilder und Icons. Für die Retina-Auflösung des iPhone bzw. iPad sind diese mit dem Zusatz `@2x` gekennzeichnet und werden dann automatisch berücksichtigt.

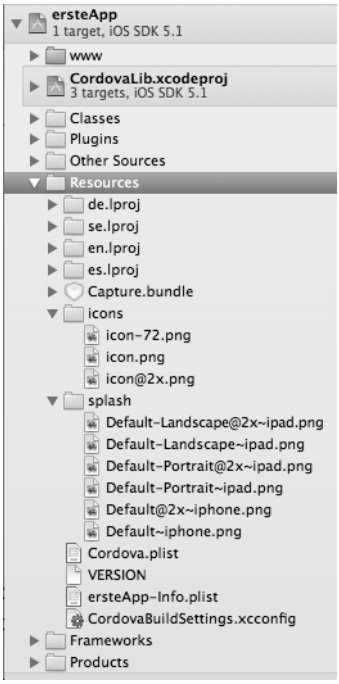

Abb. 3–14 *Ordnerstruktur für die Icons und den Startbildschirm*

Schon haben Sie eine komplette eigene App für iOS erstellt, ohne eine Zeile Objective-C zu schreiben. Das nächste System, das wir betrachten, ist Android.

3.3 Android

Um aus einer Web-App eine native Android-App zu erstellen, benötigen Sie folgende Voraussetzungen:

- Eclipse Classic[1]
- Ein aktuelles Android SDK[2]
- ADT-Plugin für Eclipse (wird im Verlauf installiert)
- PhoneGap für Android

1. *http://www.eclipse.org/downloads/moreinfo/classic.php*
2. *http://developer.android.com/sdk/index.html*