# Mobile Anwendungen mit Qooxdoo Mobile erstellen

Von Thomas Kestler, elevato GmbH – 28.11.2012

# Einleitung

Immer mehr Unternehmen wollen oder müssen Mobile Apps entwickeln. Häufig sind die Entwicklungsabteilungen darauf aber wenig vorbereitet. Erschwerend kommt hinzu, dass derzeit mindestens drei Plattformen zu berücksichtigen sind (Android, iOS, Windows Phone) und sich diese Plattformen rasant schnell weiter entwickeln.

Zunächst stellt sich die Frage, ob native Apps nötig sind oder ob man mit plattformübergreifenden Technologien wie HTML5 (und ggfs. einer Hybridlösung wie PhoneGap, jetzt ApacheCordova) zum Ziel kommt. Zwar bieten native Apps die beste Integration und Performance, aber man hat dann drei parallele Entwicklungsstränge mit drei Programmiersprachen (Java, Objective-C, C#) und Entwicklungswerkzeugen im Haus.

Deshalb haben HTML5-basierte Anwendungen den großen Charme, dass man mit einer Technologie alle Plattformen abdecken kann. (Fast) alle mobilen Browser unterstützen inzwischen HTML5 [1]. Welche Javascript-APIs der jeweilige Browser unterstützt hängt jedoch von der Plattform ab. Es gibt zahlreiche Frameworks wie z. B. jQuery Mobile oder eben Qooxdoo Mobile. Qooxdoo ist zwar noch immer relativ unbekannt, aber zu Unrecht, denn im Bereich von Web-Anwendungen (RIA) hat sich Qooxdoo[2] hervorragend bewährt. Die mobile Variante baut auf dem bewährten Framework der klassenbasierten Objektorientierung auf und bringt eine umfangreiche, durchdachte und stabile API mit.

Sofern native Features benötigt werden, die nicht bereits in HTML5 verfügbar sind, muss ein Hybrid-Ansatz mit PhoneGap (jetzt Apache Cordova[3]) genutzt werden.

# Voraussetzungen für Qooxdoo

Anfänglich wird man verwundert sein, dass man Python benötigt, um Qooxdoo-Applikationen zu bauen. Da die Installation z.B. von Active Python aber sehr einfach und problemlos ist, kommt jeder Entwickler damit schnell klar. Woran man sich anfangs gewöhnen muss ist die Unterscheidung von source-Build und build-Build. Letzterer packt alle JS-Dateien zusammen und minimiert diese. Da dies für das Debugging nicht gut geeignet ist, verwendet man während der Entwicklung den source-Build. Meist kann man einfach eine Änderung in der entsprechenden JS-Datei vornehmen und mit Reload (im Browser) die Auswirkung testen. Nur ein einigen Fällen, wenn z.B. neue Klassen benötigt werden, muss man das Generierungsscript (generate.py source) anwerfen, damit diese JS-Dateien mit übernommen werden.

Als Entwicklungswerkzeug eignet sich Eclipse sehr gut, jedoch bedingt durch die untypisierte Sprache Javascript vermisst man Syntaxchecks im Editor. So kommt es häufig vor, dass ein kleiner Tippfehler erst beim Testen auffällt. Würde man immer auf das mobile Gerät deployen, wäre das ärgerlich. Deshalb empfiehlt sich der

1 http://mobilehtml5.org/

2 http://www.qooxdoo.org

3 http://incubator.apache.org/cordova/

Einsatz eines modernen Browsers wie z.B. Chrome zum Testen vor dem Deployment auf das mobile Gerät.

# Rahmenanwendung

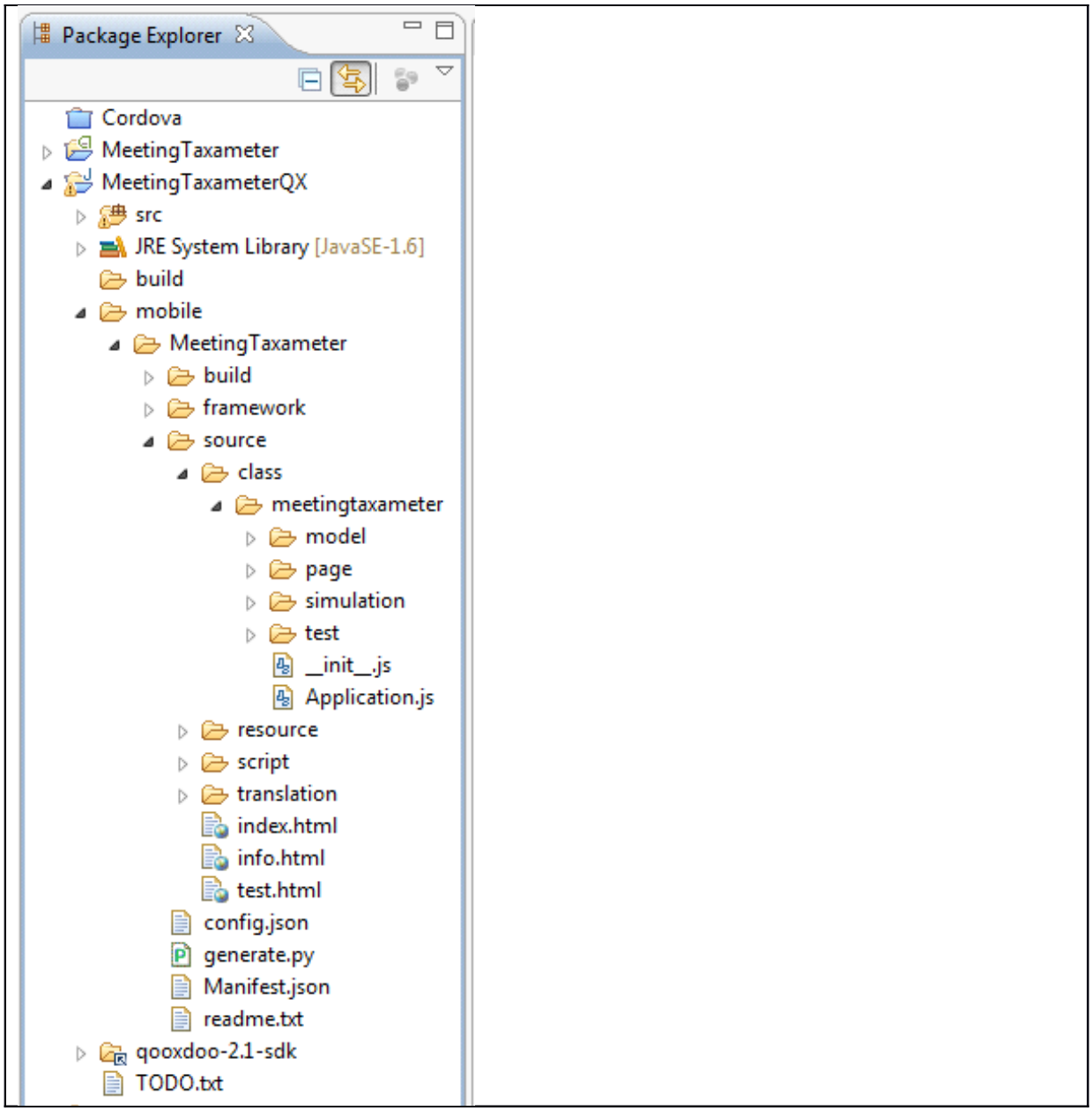
Damit die App als solche gebaut und deployt werden kann, benötigt man eine Rahmenanwendung, welche aus einer Activity besteht und ein WebView darstellt, welches die Startseite (HTML) aus den eigenen Ressourcen heraus lädt. Hier ein Beispiel für Android:

```
package de.capalogic.meetingtaxameter;

import android.app.Activity;
import android.content.res.Configuration;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;

public class MeetingTaxameterActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        WebView myWebView = (WebView) findViewById(R.id.webView1);
        WebSettings webSettings = myWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
        webSettings.setDomStorageEnabled(true);
        myWebView.loadUrl("file:///android_asset/www/index.html");
    }

    @Override
    public void onConfigurationChanged(Configuration newConfig) {
     // ignore device orientation change
     super.onConfigurationChanged(newConfig);
    }
}
```

Für iOS oder Windows Phone sieht die Rahmenanwendung ähnlich aus. Wenn PhoneGap (Cordova) ins Spiel kommt, sieht die Rahmenanwendung leicht anders aus, weil die WebView um die PhoneGap JS-API erweitert werden muss. Das kann man aber in den entsprechenden Tutorials nachlesen. Im Folgenden wird davon ausgegangen, dass die App kein PhoneGap benötigt.

# Anlegen der Qoodoo-Anwendung

Zunächst muss man ein neues Qooxdoo-Projekt anlegen, dafür stellt Qooxdoo ein entsprechendes Skript bereit. Dieses Projekt kann man dann in Eclipse einbinden. Sie dann so aus:




Als nächstes muss man eine NavigationPage ableiten, hier Input.js und diese aus Application.js heraus aufrufen. Hier der Aufruf:

```
------------------------------------------------------------------------
  Below is your actual application code...

------------------------------------------------------------------------
*/

var manager = new qx.ui.mobile.page.Manager(false);

var inputPage = new meetingtaxameter.page.Input();
manager.addDetail(inputPage);

var historyPage = new meetingtaxameter.page.History();
manager.addDetail(historyPage);
inputPage.addListener("showHistory", function(evt) {
 historyPage.show();
}, this);
```

```
   // Return to the Tweets Page
      historyPage.addListener("back", function(evt) {
        inputPage.show({reverse:true});
      }, this);



      inputPage.show();
```

Die Seite Input.js sieht ungefähr so aus:

```
qx.Class.define("meetingtaxameter.page.Input",
{
  extend : qx.ui.mobile.page.NavigationPage,

  construct : function() {
    this.base(arguments);
    this.setTitle("Dateneingabe");
  },

  members : {
   // overridden
    _initialize : function() {
      this.base(arguments);



      var form = this.__form = new qx.ui.mobile.form.Form();

      var inputName = this.__inputName = new
qx.ui.mobile.form.TextField(this.__meeting.name);
      var labelName = new qx.ui.mobile.form.Label("Name Meeting");
      labelName.setLabelFor(inputName.getId());
      form.add(inputName, "Name Meeting");
      ...
```

In der initialize-Methode werden die Widgets platziert (TextField, NumberField, Buttons, etc.)

Die Navigation von einer Seite zu nächsten erfolgt über Events und Listener. So feuert ein Button (Historie) auf der Seite Input.js einen Event `showHistory`,

```
_onTapHistory : function(evt) {
  this.fireDataEvent("showHistory", this);
},
```

der im Listener der Seite dann zum Wechsel der Seite führt:

```
inputPage.addListener("showHistory", function(evt) {
  historyPage.show();
}, this);
```

Dieser Seitenwechsel ist dabei kein Activity-Wechsel wie bei einer nativen App, sondern mittels Javascript nur simuliert. Die Anmutung ist aber sehr ähnlich zu einer nativen App.

# Stolperdrähte

Einige Stolperdrähte müssen noch entschärft werden: So müssen Javascript und ggf. DomStorage aktiviert werden und die Reaktion auf Drehen des Geräte (Device Orientation) muss berücksichtigt werden. Andernfalls wird die Seite bei Drehen einfach neu im WebView geladen und bisherige Eingaben sind verloren. Entweder man sicher den aktuellen State entsprechend (was je nach Rahmenanwendung anders abläuft) oder man konfiguriert die Rahmenanwendung entsprechend (hier Android Manifest):

```
<activity
            android:name=".MeetingTaxameterActivity"
            android:label="@string/app_name"
            android:configChanges="orientation"> <!-- API >=13: |screenSize -->
```

Häufig erlauben die Policies der Unternehmens-PCs den Anschluss der Mobilgeräte über USB nicht, was zum Entwickeln und Debuggen sehr wichtig ist. Auch werden Apple-Geräte häufig nicht unterstützt, für iOS-Entwicklung ist dies aber unerlässlich. Insgesamt hat die Entwicklung mobiler Geräte Auswirkungen auf die gesamte Infrastruktur, da für die Entwicklung mobiler Apps bestehende Policies gelockert werden müssen und auch Testumgebungen und ggf. interne App-Stores geschaffen werden müssen, um die Apps vor Veröffentlichung ausreichend testen zu können.

# Local Storage

HTML5 bietet unter anderen localStorage, um Daten lokal zu speichern. Die Speicherung obliegt dem WebView und bei Löschen der Anwendungsdaten sind diese Daten dann auch weg. Die JS-API dafür ist relative einfach: setItem („key“, „valueString“), getItem(„key“) und removeItem(„key). Man kann aber keine Objekte ablegen, sondern nur Strings, insofern bietet sich JSON-Serialisierung hierfür an. Qooxdoo bringt mit qx.lang.Json hier bereits eine gute Unterstützung mit:

```
      if (localStorage) {
   var stored = [1];
   var storedStr = localStorage.getItem("meetings");
   if (!storedStr) {
    stored[0] = meeting;
   } else {
    stored = qx.lang.Json.parse(storedStr);
    var len = stored.length;
    stored[len] = meeting;
   }
   localStorage.setItem("meetings", qx.lang.Json.stringify(stored));
  }

...

_loadList : function () {
     var arr = ["No local storage"];
       if(localStorage) {
        var str = localStorage.getItem("meetings");
        if (str) {
         arr = qx.lang.Json.parse(str, function(key, value) {
          if(key == "start" || key == "end") {
           return(new Date(Date.parse(value)));
          }
          return value;
         });
        }
       }
       this.__list.setModel(new qx.data.Array(arr));
    },
```

Die Konvertierungsfunktion in parse() wandelt Datums-Strings wieder in Date-Objekte zurück.

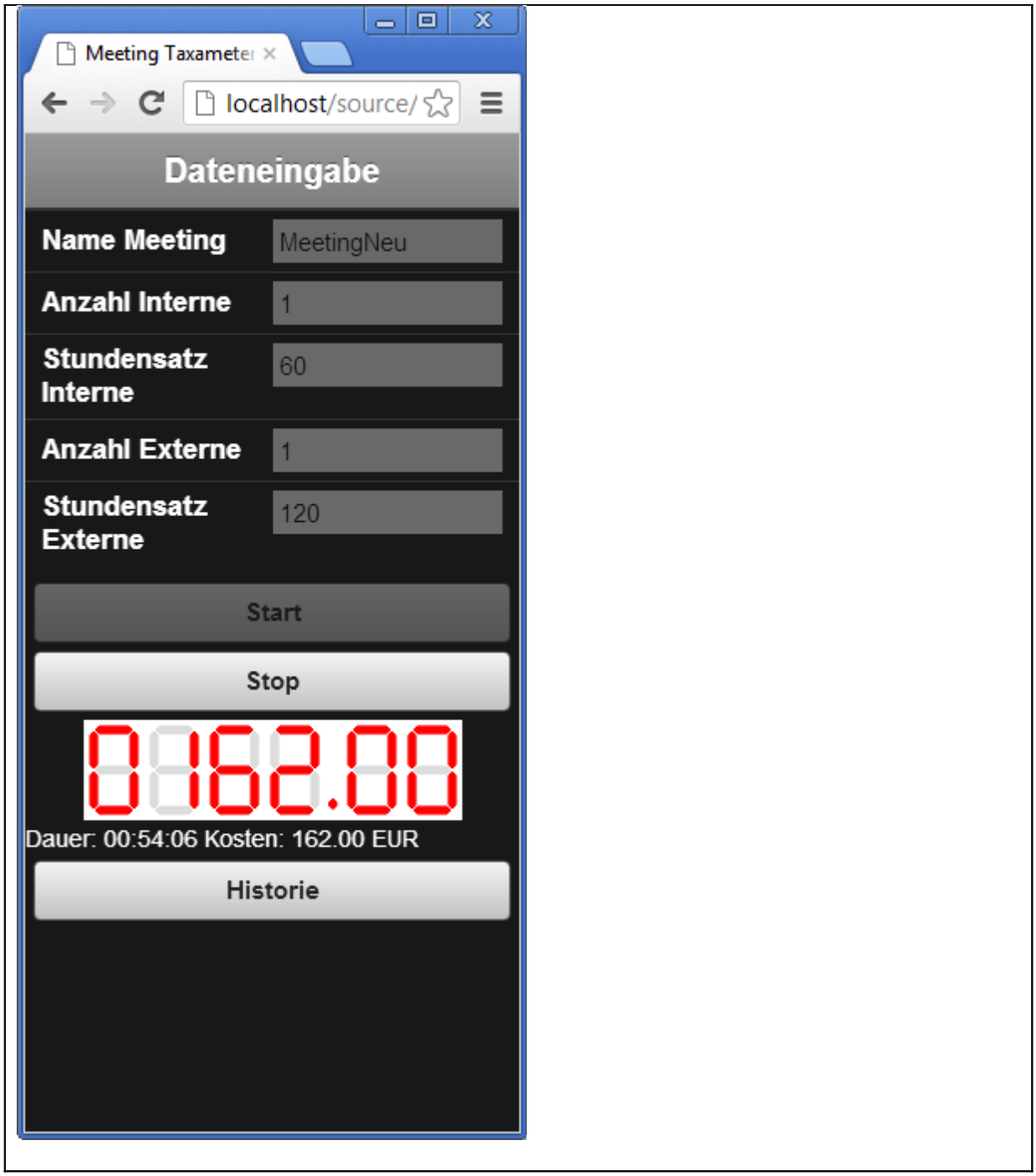
# Testen im Browser

Während der Entwicklung ist es unerlässlich schnell Testergebnisse zu haben. Zum Glück klappt das mit einem Browser wie dem Chrome und einem Web-Server, der die Inhalte ausliefert, sehr einfach. Auf den Web-Server kann man auch verzichten, wenn man im Browser Ausführen von Javascript über file:-Protokoll zulässt (Firefox ermöglicht dies, Chrome wohl auch). Aber mit NANOHttpd im Eclipse-Projekt hat man ganz schnell einen Webserver parat und muss nichts an der Browser-Konfiguration ändern. Einziger Stolperstrick: man muss config.json etwas anpassen und die generierte JS-Datei nach jedem generate-Lauf anpassen

```
var libinfo = {"__out__":{"sourceUri":"script"},"meetingtaxameter":
{"resourceUri":"../source/resource","sourceUri":"../source/class"},"qx":
{"resourceUri":"..//framework/source/resource","sourceUri":"../framework/so
urce/class","sourceViewUri":"https://github.com/qooxdoo/qooxdoo/blob/%
{qxGitBranch}/framework/source/class/%{classFilePath}#L%{lineNumber}"}};
```

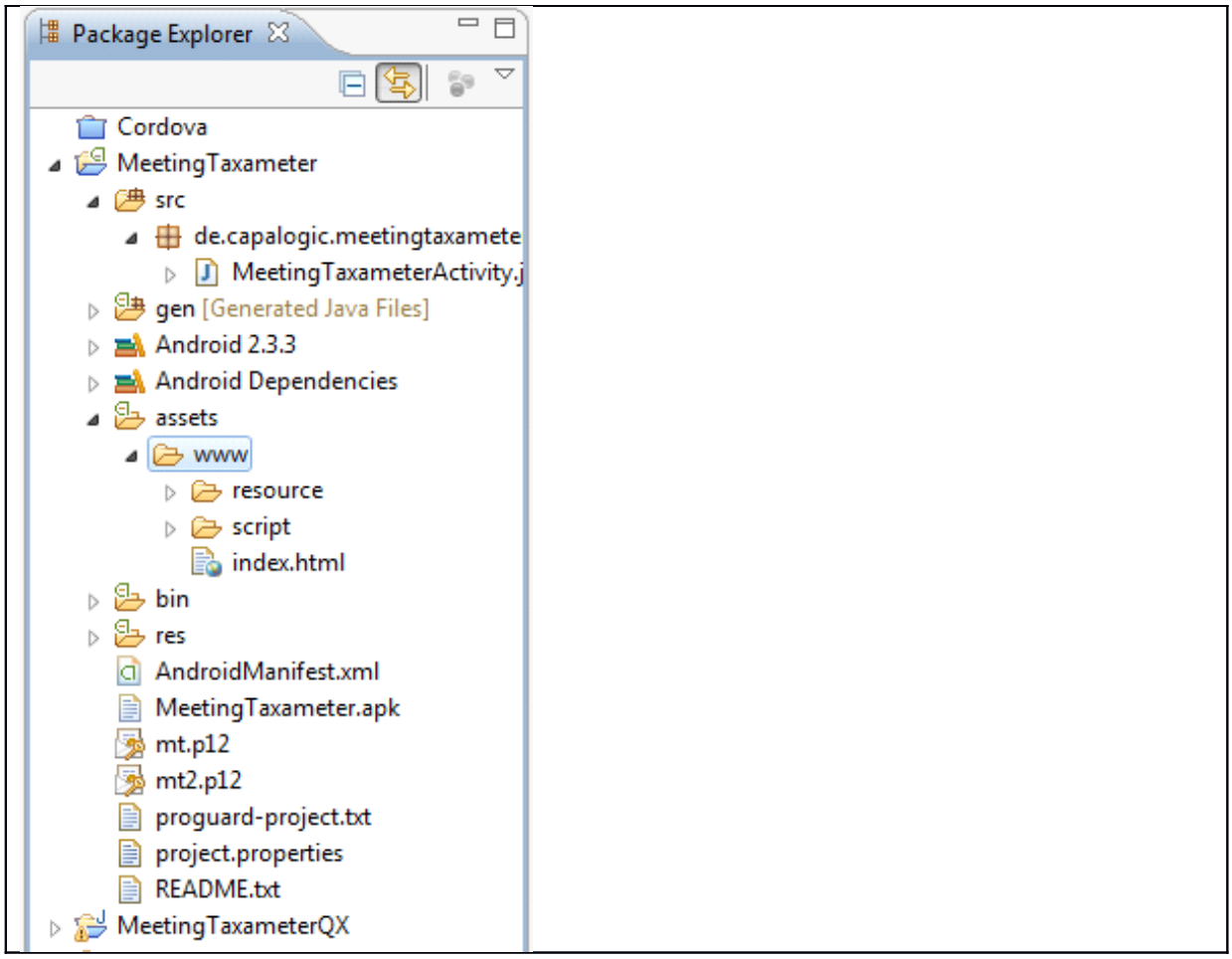
In gelb sind die Stellen markiert, die man manuell (oder per Skript nach jedem Generate-Lauf anpassen muss. Dazu muss zudem der Ordner `framework` aus dem Qooxdoo-SDK in das Eclipse-Projekt kopiert werden, nicht ganz elegant, aber pragmatisch. Es gibt sicher noch viele  weitere Wege, das Testen im Browser zu ermöglichen.

Entscheidend ist, dass man die mobile App weitestgehend im Browser testen kann.




## Übernahme in Rahmenanwendung

Wenn die Anwendung im Browser soweit läuft, können die nötigen Dateien in die Rahmenanwendung übernommen werden. Dazu müssen die `index.html` und die Ordner `resource` und `script` in den Ordner `assets/www` kopiert werden:




Wichtig zu wissen ist nur noch, dass die Dateien über folgende URL in die WebView geladen werden müssen:

```
file:///android_asset/www/index.html
```

# PhoneGap

Für Zugriffe auf native Features kann PhoneGap (Cordova) genutzt werden. Apache Cordova wird inzwischen aber nur noch als Source-Code ausgeliefert und der Entwickler muss sich die Bibliotheken selbst bauen. Dies ist letztlich den rasend schnellen Release-Zyklen der mobilen Plattformen geschuldet, macht den Einsatz von Cordova aber umständlicher. Am einfachsten geht es für Android, sofern man den passenden API-Level im Android SDK geladen hat (z. B. Cordova 2.2.0 benötigt Android-17). Dass man für iOS-Entwicklung einen Apple-Rechner benötigt, wird niemand mehr überraschen. Zum Bauen von Cordova für iOS benötigt man diesen Rechner ebenfalls.

# Fazit

Qooxdoo Mobile bietet eine ausgereifte API zur Erstellung von plattformübergreifenden Apps, vergleichbar mit jQuery Mobile und anderen. Die Qooxdoo API hat aber den Vorteil, dass sie sehr gut durchdacht und umgesetzt ist. Mobile Apps können mit Qooxdoo nach kurzer Eingewöhnungszeit sehr effizient erstellt werden. Sofern native Features mit PhoneGap genutzt werden sollen, steigt der Aufwand, da das Bauen von PhoneGap/Cordova für die unterschiedlichen Plattformen relativ aufwändig ist.

---

Thomas Kestler ist Geschäftsführer der elevato GmbH. http://www.elevato.de