QooXdoo – Javascript Framework

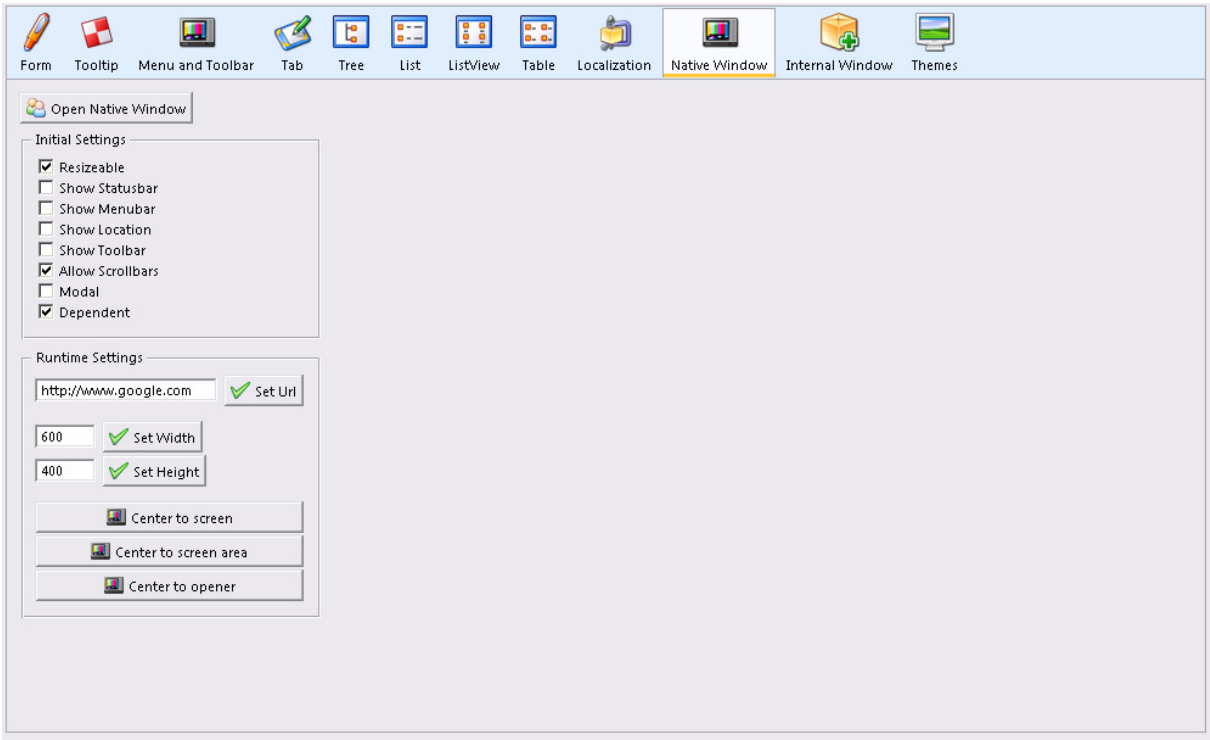
Was es ist

Qooxdoo ist ein Framework welches komplett mit Javascript programmiert ist. Dadurch dass es Javascript ist, wird es später im Browser ausgeführt. Qooxdoo dient dazu RIA (Rich Internet Applications) zu entwickeln. Google-Maps oder Yahoo's Email Programm sind Javascript-Anwendungen mit einem Backendserver. Qooxdoo beschränkt sich darauf Windows ähnliche GUI's darzustellen.

Das Framework gehört also genau zu den beliebten Buzzwörtern Web 2.0, Ajax usw.

Qooxdoo spricht man in etwa wie „Kuggs Du" oder Englisch „cooks do".

Wer erstmal einen Eindruck bekommen will, sollte sich die Demos unter: http://qooxdoo.org/demo anschauen.



QooXdoo Showcasedemo

Das Framework hat jedoch eine Besonderheit. Es wird eine „Make"-Umgebung gebracht. Diese lässt sich in Windowssystemen mit Cygwin herstellen und unter Linux sollte ein „Make" schon vorhanden sein.

Man entwickelt zuerst mit „unkompilierten"-Klassen. Durch einen „make build" wird dann die Anwendung optimiert. Bei der Optimierung werden z.B. ungenutzte Klassen und Ressourcen entfernt. Dadurch kann dann die fertige Anwendung wesentlich schneller auf dem Zielsystem im Browser ausgeführt werden.

Für den ganzen Aufwand erhält man allerdings ein schneller und sehr funktionelles Framework. Das Framework kann selbstverständlich auch durch Backendprogramme mit Webservern/Datenbanken verbunden werden. Im Downloadbereich von den Qooxdoo Webseite finden sich Beispiele u.a. für PHP Backendprogramme.

Jetzt geht es los!

Als erstes wird das SDK benötigt. Dieses lässt sich hier runter laden:
http://qooxdoo.org/download

Nach dem entpacken muss als erstes ein Dev Source erzeugt werden. Dazu wechselt man in den Ordner „Frontend" und ruft „Make [source]" auf. Das erzeugt dann den Dev Source.

```
Wget http://mesh.dl.sourceforge.net/sourceforge/qooxdoo/qooxdoo-0.6.5-sdk.tar.gz
tar -xzf qooxdoo-0.6.5-sdk.tar.gz
cd qooxdoo-0.6.5-sdk/frontend/
make
```

Dieser kann getestet werden wenn man folgende HTML-Datei aufruft:
frontend/application/index.html

Anschließend sollte man sich ein Skelett generieren lassen, welches dann als Grundlage zum entwickeln dient. In dem Ordner „frontend/application" liegt eine Zip/Tar Datei. Diese einfach in einen neuen Ordner entpacken. Anschließend einfach in den Ordner wechseln und erneut „Make" aufrufen. Das Make wird allerdings erstmal meckern. In der Makefile müssen erst 2 Variablen gesetzt werden.

```
#
# Location of your qooxdoo distribution
# Could be relative from this location or absolute
#
QOOXDOO_PATH = PLEASE_DEFINE_QOOXDOO_PATH

#
# The same as above, but from the webserver point of view
# Starting point is the application HTML file of the source folder.
# In most cases just add a "/.." compared to above
#
QOOXDOO_URI = PLEASE_DEFINE_QOOXDOO_URI
```

Wenn die Pfade in die Variablen QOOXDOO_PATH und QOOXDOO_URI angepasst sind, wird das „Make" wieder aufgerufen. Nun wird das Skelett vorbereitet. Nach erfolgreichen vorbereiten, liegt im Verzeichnis „source\class\custom" eine Datei „Application.js". Diese enthält den Programmcode zu unserer Anwendung.

```
cp skeleton.tar.gz ../../../   #aus dem “frontend Verzeichnis aus”
cd ../../../
tar -xzf skeleton.tar.gz
cd skeleton
>>Makefile angepasst
>>QOOXDOO_PATH = ../qooxdoo-0.6.5-sdk
>>QOOXDOO_URI = ../../qooxdoo-0.6.5-sdk
make
```

Getestet kann die Anwendung durch den Aufruf der „index.html" im „source" Verzeichnis. Das Ergebnis sollte dann nach (längerer) Ladezeit wie folgt aussehen:

Die längere Ladezeit ist Aufgrund der DEV Version vom Skelett. Dieses lädt bei Aufruf über 300 Dateien nach. Später in der Build-Version ist das dann nicht mehr der Fall, da diese dann optimiert wurde. Das Nachladen beschränkt sich dann auf ca. 5 Dateien.

Anpassen des Skeletts

Ich nutze die Eclipse IDE mit dem Aptana Plugin zur Entwicklung. Mit den Eclipse lade ich nun die Datei „skeleton/source/class/custom/Application.js". Die Syntaxunterstützung habe ich leider nicht zum laufen bekommen. Das liegt an dem Qooxdoo, da es keine qx,js mehr gibt und diese auch nicht mehr generiert werden kann.
(http://qooxdoo.org/documentation/related_projects/aptana_ide)

Update:
Inzwischen nutze ich die Betaversion von dem Adobe JSclipse Plugin. Dieses hat etwas mehr Unterstützung für die Objektorientierten-Programmierung unter Javascript.

In dem Skelett gibt es in Zeile 43 folgenden Codeabschnitt:

```
qx.Proto.main = function(e)
{
  […]
}
```

Das ist ein z.B. in der Programmiersprache C die Hauptfunktion, welche bei Webseiten „Start" aufgerufen wird. Das ist der Punkt wo wir anfangen die Klasse zu erweitern. In der Main werden alle Initialisierungen durchgeführt. Das Framework selbst lässt sich wie jede Objektorientiert-GUI-Programmiersprache programmieren.
Fügen wir folgenden Code der Main hinzu erhalten wir ein Label welches auf der Position x100, y200 angezeigt wird. Der Text des Labels ist dann: „Gib was ein:"

```
var Label1 = new qx.ui.basic.Label('Gib was ein:', '');
Label1.setTop(100);
Label1.setLeft(200);
Label1.addToDocument();
```

Da wir jetzt schon ein schönes Label haben, könnten wir noch prima ein passendes Eingabefeld und einen Knopf hinzu bauen.

```
var textfield1 = new qx.ui.form.TextField('Muster');
textfield1.setTop(116);
textfield1.setLeft(200);
textfield1.setWidth(300);
textfield1.addToDocument();

var buttonsend = new qx.ui.form.Button("Absenden");
buttonsend.setTop(116);
buttonsend.setLeft(504);
buttonsend.setHeight(20);
buttonsend.addToDocument();
```

Das wir jetzt eine Artsuchmaske haben, die wie folgt aussieht, brauchen wir nun nur noch ein Steuerelement welches die Anzeige unserer Suche übernimmt.

Die Trefferanzeige möchte ich in diesem Beispiel eine „Table" nutzen. Diese lässt sich wie folgt erzeugen:

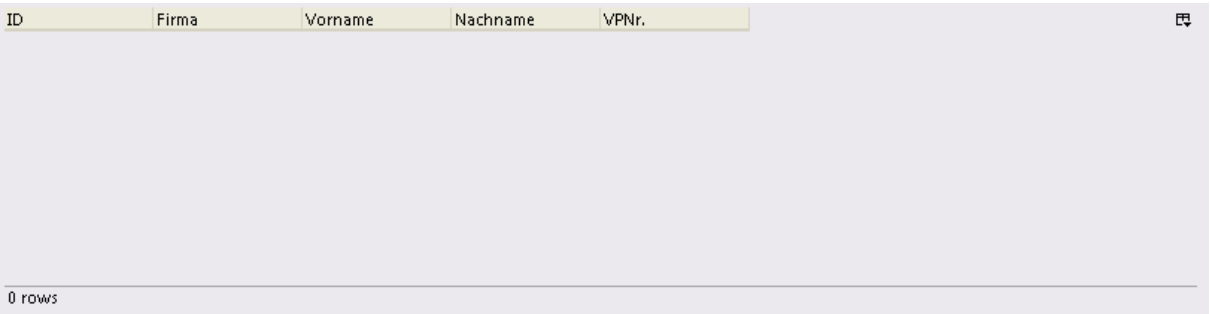
```
var table = new qx.ui.table.Table(tableModel);
table.setTop(150);
table.setLeft(200);
table.setWidth(800);
table.setHeight(200);
table.addToDocument();
```

Der Parameter „tableModel" in Constructor des Tableobjekts dient dazu um der Table mitzuteilen wie viele Spalte sie hat. Über dieses Objekt werden noch weitere Einstellungen wie z.B. Spaltennamen vorgegeben.

Das TableModel erzeuge ich wie folgt:

```
  var tableModel = new qx.ui.table.SimpleTableModel();
  tableModel.setColumns([ "ID", "Firma", "Vorname", "Nachname", "VPNr." ]);
```

Die Tabelle sollte nun so aussehen:



Ziemlich hässlich finde ich. Daher passt folgender Code die Tabellenoptik noch ein wenig an.

```
  with (table) {
    set({ border:qx.renderer.border.BorderPresets.getInstance().thinInset });
    getSelectionModel().setSelectionMode(qx.ui.table.SelectionModel.SINGLE_SELECTION);
    setColumnWidth(0, 80);
    setColumnWidth(1, 200);
    setColumnWidth(2, 200);
    setColumnWidth(3, 200);
    setColumnWidth(4, 50);
  };
```

Nun müssen wir nur noch die Musik einbauen. Bei Knopfdruck überprüfen wir ob überhaupt etwas eingegeben wurde. Wenn das der Fall war machen wir eine Ajaxanfrage an ein PHP-Backendscript. Anschließend zeigen wir das Ergebnis in der Table an. Schwierig ? NEIN!

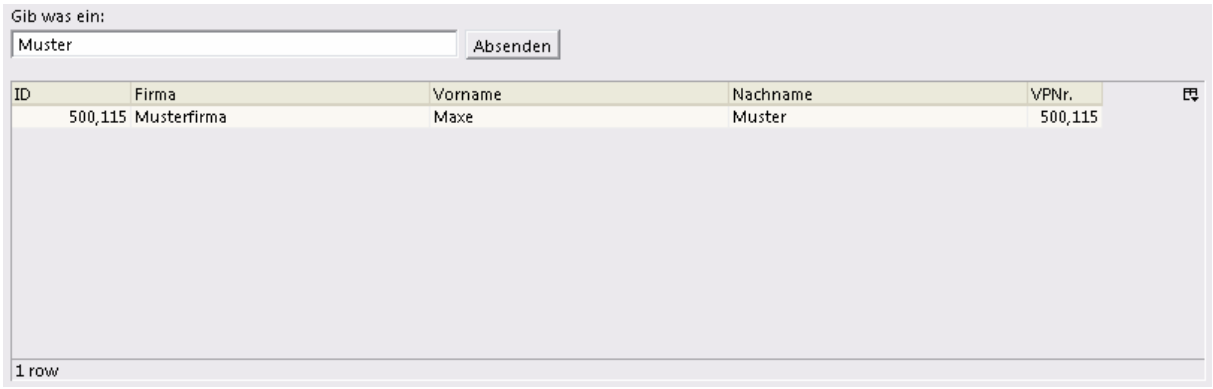
```
  buttonsend.addEventListener("execute", function(e) {
    if (textfield1.getValue() == "") {
      alert('Es muss etwas eingegeben werden.');
    } else {

      var req = new qx.io.remote.Request(
         "http://localhost/qooxdoo/php/test.php",
         "GET",
        "text/plain");

      req.addEventListener("completed", function(e) {
        var content = e.getData().getContent();
        var tabledata = qx.io.Json.parse(content);
        tableModel.setData(tabledata);
      });
```

```
      req.setParameter("suche", textfield1.getValue());
      req.send();
    }
  });
```

Final sollte die Anwendung nun so aussehen, nachdem der „Absenden"-Knopf gedrückt wurde.



Als Backend PHP

Meine Umgebung besteht aus einen MS-SQL 2000 Server, Apache Webserver mit installierten PHP 5.2.0. Das alles läuft auf Microsoft Windows XP (32Bit). Für das Backendscript habe ich mir ausgedacht, dass ein „Get" mit „Suche" Parameter in einer Kundenbank sucht und ein JSON Datensatz ausgibt. Dazu initialisiere ich eine PEAR-DB Verbindung zu meinen MSSQL. Dann führe ich anschließend das Query mit dem Parameter aus. Die Funktion „getAll" liefert mir ein Array of Array mit den Daten. Diesen Output wandele ich dann anschließend in UTF-8 um. Das ist notwendig, da die „JSON_encode" Funktion nur UTF-8 versteht. Meine Datenbank ist altmodisch mit ISO 8859-1 gefüllt.

```
<?php
  // Dieser PHP CODE dient zu Präsentationszwecken,
  // daher ist er für Produktionszwecke Aufgrund fehlender
  // Sicherheitsroutinen nicht zu empfehlen!
  $_CONFIG['Datenbank'] = "mssql://user:pass@server/DB";
  $_CONFIG['ROOT_PATH'] = "E:\\work\\qooxdoo\\php\\";

  $include_path = ini_get('include_path');
  $include_path = $_CONFIG['ROOT_PATH'] . 'lib/PEAR/' . PATH_SEPARATOR . $include_path;
  ini_alter('include_path', $include_path);

  require_once('./lib/PEAR/DB.php');

  $datenbank = &DB::connect($_CONFIG["Datenbank"]);
  if (DB::isError($datenbank)) {
   die($datenbank->getMessage());
  }

  $sql = "select top 500 INT_KONTAKT, STR_FIRMENNAME, STR_VORNAME, STR_NACHNAME,
    INT_KONTAKT from tbl_kontakte
    where (STR_FIRMENNAME like ?) or (STR_VORNAME like ?) or (STR_NACHNAME like ?)";

  $suche = isset($_REQUEST["suche"]) ? utf8_decode($_REQUEST["suche"]) : "";
  $p = array($suche, $suche, $suche);
  $recordset=$datenbank->getAll($sql, $p, DB_FETCHMODE_ORDERED); // DB_FETCHMODE_ASSOC
  if (DB::isError($recordset)) {
    die($datenbank->getMessage());
  }

  // Das JSON verarbeitet NUR UTF8 - meine DB hält aber ISO 8859-1
  // Daher kodiere ich alles ausser Nummeric in UTF8 um.
  function AnsiToUTF8(&$item, $key) {
    if (!is_array($item)) {
      if (!is_numeric($item)) {
        $item = utf8_encode($item);
      }
    }
  }

  // nun Umkodieren
  array_walk_recursive($recordset, 'AnsiToUTF8');
  // JSON erzeugen
  echo json_encode($recordset);
?>
```

Das erzeugt das PHP-Script:

```
[[500115,"Musterfirma","Maxe","Muster",500115]]
```

Zusehen ist ein JSON Array welches ein Array mit 5 Feldern enthält. Diese Daten können dann direkt im Qooxdoo weiterverarbeitet werden.

Aussichten

Qooxdoo wird wohl Ende März als Version 0.7 erscheinen. Dort wurde noch mal stark am Objektorientiertendesign gearbeitet. Die Klassen sollen dann besser lesbar und leichter verwendbar sein.
Eine Übersicht findet sich hier:
http://qooxdoo.org/documentation/articles/comparison_of_class_declarations_between_0.6.x_and_0.7

Über den Autor

Bernd Ott ist Jahrgang 1975 und seit Mitte der achtziger Jahre ist er im Bereich der Softwareentwicklung tätig. Er arbeitet seit 1995 als festangestellter Softwareentwickler
mit den Programmiersprachen Delphi, PHP, Java und C. Seit 1999 betreibt er nebenberuflich ein Einzelunternehmen. Dieses beschäftigt sich mit Unternehmensberatung, Prozesstechnik, Lichttechnik und DTP. In seiner verbleibenden Freizeit ist er Diskjockey (house, techno musik).

Links

http://qooxdoo.org/ - Das JS Framework
http://www.php.net/ - PHP als Backend
http://pear.php.net/ - Hilfeklassen für PHP
http://www.json.org/ - JavaScript Object Notation
http://www.eclipse.org/ - Freie IDE
http://www.eclipse.org/php/ - PHP Plugin für die IDE
http://www.aptana.com/ - JS Plugin für die IDE