Universität Hamburg

Hausarbeit im Seminar:

„Android: Plattform für Mobile Geräte“

(Google) Gears

Marina Eins
4eins@informatik.uni-hamburg.de
Studiengang Diplom Informatik
Matr.-Nr. 5703259
Fachsemester 11

Betreuer: Prof. Dr. Thomas Ludwig
Julian Kunkel / Timo Minartz / Michael Kuhn

Inhaltsverzeichnis

1. Einleitung

Beim Surfen im Internet sind Webapplikationen nicht mehr wegzudenken, bringt ihre Verwendung doch unbestreitbare Vorteile mit sich.
Lediglich ein Web-Browser mit Internetzugang wird beim Benutzer vorausgesetzt, wodurch sich Schwierigkeiten im Bezug auf Hardware- und Software-Voraussetzungen minimieren lassen.
Auch muss für die Verwendung von Web-Applikationen, bis auf hin und wieder vielleicht mal ein Browser-Plugin (z.B. Flash), nichts installiert werden.
Die neuesten Features einer Anwendung stehen ohne komplizierte Updates zur Verfügung.
Der große Nachteil von Webapplikationen liegt jedoch gerade in diesen scheinbar geringen Voraussetzungen für ihre Nutzung verborgen.
Ein Web-Browser mit Internetzugang ist zwar häufig, jedoch nicht immer in dieser Kombination verfügbar.
Gerade bei Reisen mit der Bahn oder im Flugzeug, kann es des öfteren zu langsamen und instabilen oder gar fehlenden Internetverbindungen kommen.
Die Webanwendungen, die dann versucht werden im Offline-Modus ausgeführt zu werden, lassen sich nicht mehr über den Browser laden.
Eine besonders unbequeme Situation z.B. bei Verwendern von Weboffice-Anwendungen wie „Google Kalender“, die durch die fehlende Internetverbindung plötzlich keinen Zugriff mehr auf ihre geplanten Termine haben sollen.
Um diese Problematik zu lösen, hat sich das US-Unternehmen „Google Inc.“ daher an die Entwicklung der Software „Google Gears“ gemacht, die in dieser Seminararbeit näher beleuchtet und deren Verwendung und Implementierung für statische als auch dynamische Webanwendungen vorgestellt werden soll.
Auf die Android-Plattform wird dabei in so fern eingegangen, als dass speziell für Android die Entwicklung einer dynamischen Beispiel-Anwendung Schritt für Schritt vorgestellt wird.

2. Gears Allgemein

Um zu klären wobei es sich bei Google Gears eigentlich handelt und was genau ein User benötigt um es zu verwenden, sei im folgenden ein allgemeiner Überblick über Definition, Hintergrund und Voraussetzungen von Google Gears vorgestellt.

2.1 Definition

Abbildung 1: Google Gears - Logo

Bei Google Gears (dt. „Getriebe“) handelt es sich um eine Open-Source-Software, die von Google Inc. unter BSD-Lizenz veröffentlicht wurde.
Google Gears stellt für den Benutzer ein Browser-Plugin dar, das den Web-Browser dahingehend erweitert, dass Online-Webanwendungen auch im Offline-Betrieb genutzt werden können.
Dadurch soll zum einen die Nutzung von Web-Anwendungen zu jeder Zeit (offline) unterstützt werden und zum anderen auch deren Performance verbessert werden.

2.2 Hintergrund

Vorgestellt wurde Google Gears erstmalig im Mai 2007 während eines Vortrages auf dem Google Developer Day 2007 in Sydney.
Der Original-Vortrag von Aaron Boodman kann über „YouTube“ online angesehen werden.
(`http://www.youtube.com/watch?v=cQyha30nm6k`)

Seit Juni 2008 läuft das „Google Gears“ - Projekt allerdings nur noch unter dem Namen „Gears“.
Die Idee dahinter sollte das Signalisieren der Code-Freiheit von Gears sein.
Das Unternehmen Google wollte somit verstärkt darauf hinweisen, dass Gears ein offenes Projekt ist und nicht nur vorn Google betrieben wird.
Im Bezug darauf folgte dann auch im Juni 2008 die entsprechende Logo-Änderung.

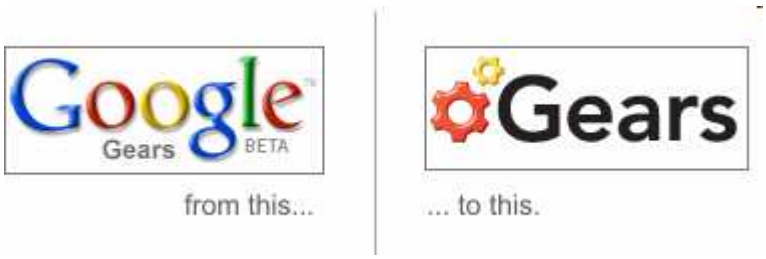


Abbildung 2: Altes und Neues Gears-Logo

2.3 Voraussetzungen

Damit Gears in entsprechenden Webanwendungen verwendet werden kann bedarf es zweierlei Voraussetzungen für den Benutzer.
Er benötigt zum einen das Gears-Plugin, das er über seinen Web-Browser runterladen und installieren muß und zum anderen eine Gears-kompatible Web-Anwendung, die er über Gears offline benutzen können möchte.

Das Gears-Plugin ist auf der Gears -Homepage (`http://gears.google.com`) kostenlos erhältlich. Die Kompatibilität des Plugins beschränkt sich jedoch auf folgende Browser:

- Firefox 1.5+ für Windows, Mac OS X, Linux
- Internet Explorer 6.0+ für Windows
- Internet Explorer Mobile 4.01+ für Windows Mobile 5+
- Safari 3.1.1+ für Mac OS X

Für Android-Browser ist es standardmäßig schon auf allen Android-Geräten integriert.

Was Gears-kompatible Seiten betrifft seien folgende Anwendungen als Beispiele genannt:

- Google Mail (E-Mail-Dienst)
- Google Kalender (Webkalender)
- Remember the milk (Aufgabenverwaltung)
- Youtube (Videoupload)
- Studivz (Fotoupload)

Generell lässt sich jedoch jede Webanwendung mit ein paar Änderungen Gears-fähig erstellen, worauf im Kapitel 3.4 und 4.2 näher eingegangen wird.

3. Gears im Detail

Um zu klären, aus welchen Komponenten sich Gears genau zusammensetzt und inwiefern diese für das Erstellen von offline-fähigen Webanwendungen von Nöten sind, sei im Folgenden anhand von Komponentendefinitionen, dem architektonischen Aufbau von Gears-Anwendungen und einem Beispiel für die Entwicklung einer statischen Gears-Webanwendung gezeigt.

3.1 Komponenten

Die Gears-Software besteht im Wesentlichen aus 2 Komponenten.
Zum einen aus dem Gears-Browser-Plugin und zum anderen aus einer Sammlung von JavaScript-APIs.
Das Gears-Plugin ermöglicht den APIs den Zugriff auf den lokalen Datenträger und die JavaScript-APIs wiederum ermöglichen Webanwendungen Offline-Funktionalitäten zur Verfügung zu stellen.
Ansprechen lassen sich die APIs per Javascript und unterteilt sind sie in 9 Module.
Im Folgenden eine grobe Übersicht dieser Module:

- **Factory**: dient zum Instantiieren aller anderen Gears-Objekte.
- **LocalServer:** dient dem lokalen Speichern von Anwendungs-Ressourcen
- **Database:** dient lokalem Speichern von Anwendungs-Daten in einer Datenbank
- **WorkerPool:** dient dem *asynchronen* (im Hintergrund) ausführen von Javascript
- **Geolocation:** dient der Standortbestimmung des Clients
- **Desktop:** dient dem Anlegen von Verknüpfungen auf dem Desktop
- **HttpRequest:** ermöglicht HTTP-Anfragen
- **Timer**: kann Zeitschalter setzen sowie auf sie reagieren
- **Blob**: ist eine Javascript-Klasse für Binärdaten

Übersicht aller APIs und des Source-Codes unter: http://code.google.com/p/gears/

Das Factory-Modul

Die Factory-Klasse dient zum instantiieren aller anderen Gears-Objekte.
Um sie zu definieren muß in jeder Gears-Anwendung das JavaScript „gears_init.js“ implementiert werden, das zur Gears-Distribution gehört und auf der Gears-Homepage erhältlich ist. Über eine „create“-Methode ist die Factory in der Lage ein Objekt der jeweils übergebenen Klasse anzulegen.

Beispiel: Anlegen eines Datenbank-Objektes

```
<script type="text/javascript" src="gears_init.js">
</script>
<script type="text/javascript">
//prüfen ob Gears installiert ist
if (window.google && google.gears)
//über create() ein Database-Objekt instantiieren
{ var db = google.gears.factory.create('beta.database'); db.open(); }
</script>
```

Das LocalServer-Modul

Die LocalServer-Klasse ermöglicht es einer Webanwendung ihre Anwendungs-Ressourcen (HTML, CSS, JavaScripte, Bilder etc.) in einem lokalen Store zu cachen. Dadurch wird Offline-Funktionalität ermöglicht.
Für das Caching der Anwendungs-Ressourcen gibt es 2 verschiedene Stores:
- „ResourceStore“: für manuelles Caching der Ressourcen über Javascript
- „ManagedResourceStore“: für manuelles & automatisches Caching der Ressourcen über ein Manifest-File.

Das Manifest-File ist dabei ein TXT-File, das alle URLs der zu cachenden Anwendungs-Ressourcen auflistet. Die im Manifest-File gelisteten Files können manuell über die „checkForUpdate()“-Methode der ManagedResourceStore-Klasse und automatisch bei jeder Anfrage nach Ressourcen aus dem ManagedResourceStore gecached bzw. geupdatet werden.

Beispiel: Cachen von Files in einen ManagedResourceStore

```
//gears_init.js einbinden
<script type="text/javascript„ src="gears_init.js"></script>
<script type="text/javascript">
// LocalServer-Objekt anlegen
var localServer = google.gears.factory.create('beta.localserver');
// ManagedResourceStore über den LocalServer anlegen
var store = localServer.createManagedStore('test-store');
// Dem ManagedResourceStore sein Manifest-File zuweisen
store.manifestUrl = 'site-manifest.txt';
// Files aus dem Manifest-File cachen bzw. updaten
store.checkForUpdate();
</script>
// Inhalt von „site-manifest.txt“
// Version des Manifest-File-Formats (1 oder 2)
{ "betaManifestVersion": 1,
// String, der die Version der Ressourcen-Sammlung angibt
"version": "version 1.0",
// Die zu cachenden Ressourcen
"entries": [ { "url": "site.html" }, { "url": "gears_init.js" } ] }
```

Das Database-Modul

Die Database-Klasse dient zum lokalen Speichern von Anwendungs-Daten in einer SQLite-Datenbank (relationale SQLfähige DB).
Die Daten werden über SQL-Statements in einer „execute“-Methode gespeichert, abgerufen und geändert.

Beispiel: Speichern und Abrufen von Daten in ein einer Datenbank

```
<script type="text/javascript" src="gears_init.js"></script>
<script type="text/javascript">
// Database anlegen
var db = google.gears.factory.create('beta.database');
db.open('database-test');
// Anlegen einer Tabelle „Test“ in der Datenbank
db.execute('create table if not exists Test (Phrase text, Timestamp int)');
// Test-Tabelle befüllen
db.execute('insert into Test values (?, ?)',
['Monkey!', new Date().getTime()]);
// Ruft Daten aus der Test-Tabelle ab und gibt das ResultSet jeder
// Zeile durch „@“ verbunden aus
var rs = db.execute('select * from Test order by Timestamp desc');
while (rs.isValidRow()) { alert(rs.field(0) + '@' + rs.field(1));
rs.next(); }
rs.close();
</script>
```

Das WorkerPool-Modul

Die WorkerPool-Klasse ermöglicht Web-Anwendungen das Ausführen von JavaScript-Code (bzw. Child-Workern) im Hintergrund, ohne dass die Ausführung des Hauptseiten-Script (bzw. Parent-Worker) geblockt wird.
Worker kommunizieren miteinander nur über das Senden von Message-Objekten.

Beispiel: WorkerPool anlegen und Worker miteinander kommunizieren lassen

```
// main.js (bzw. parent-worker)
// WorkerPool anlegen
var workerPool = google.gears.factory.create('beta.workerpool,);
// wenn Message im WorkerPool eingeht folgt ein Alert
workerPool.onmessage = function(a, b, message) {
alert('Received message from worker ' + message.sender + ': \n' +
message.body); };
// WorkerPool mit JS-Code aus einer URL anlegen
var childWorkerId = workerPool.createWorkerFromUrl('worker.js');
// Message an Child-Worker senden
workerPool.sendMessage(["3..2..", 1, {helloWorld: "Hello world!"}],
childWorkerId);

// worker.js (bzw. child-worker)
var wp = google.gears.workerPool;
// Wenn Message im Child-Worker eingeht, Antwort erzeugen
wp.onmessage = function(a, b, message) {
var reply =
message.body[0] + message.body[1] + "... " + message.body[2].helloWorld;
// Antwort an den Sender senden
wp.sendMessage(reply, message.sender); }
```

Das Geolocation-Modul

Die Geolocation-Klasse ermöglicht einer Web-Anwendung die geographische Position eines Nutzers zu bestimmen und zu ermitteln, wenn sich seine Position ändert.
Die Positionsbestimmung erfolgt anhand von GPS, WLAN oder Sendemasten über deren Funkstärke die Position errechnet wird.

Beispiel: Längen- und Breitengrad der aktuellen Position ermitteln

```
// Geolocation-Objekt anlegen
var geo = google.gears.factory.create('beta.geolocation');
// Über „updatePosition()“ Längen- und Breitengrad abrufen und ausgeben
function updatePosition(position) {
alert('Current lat/lon is: ' + position.latitude + ',' +
position.longitude); }
// Über „handleError()“ Fehlermeldung ausgeben
function handleError(positionError) {
alert('Attempt to get location failed: ' + positionError.message); }
// Über die Geolocation-Methode „getCurrentPosition()“ die
// Callback-Funktionen für erfolgreiche oder nicht erfolgreiche
// Positionsbestimmung angeben
geo.getCurrentPosition(updatePosition, handleError);
```

Das Desktop-Modul

Die Desktop-Klasse ermöglicht den Zugriff auf Desktop-bezogene Funktionalitäten, wie das Anlegen eines Shortcuts auf dem User-Desktop oder die benutzergesteuerte Auswahl lokaler Files für die Verwendung in einer Web-Anwendung.

Beispiel: Anlegen eines Shortcuts und Auswahl lokaler Files

```
// Desktop-Interface anlegen
var desktop = google.gears.factory.create('beta.desktop');
// Desktop-Shortcut erzeugen (Übergabe von Name, URL, Icons, Beschreibung)
desktop.createShortcut(
'Test Application',
'http://example.com/index.html',
{'128x128': 'http://example.com/icon128x128.png', '48x48':
'http://example.com/icon48x48.png',
'32x32': 'http://example.com/icon32x32.png', '16x16':
'http://example.com/icon16x16.png'},
'An application at http://example.com/index.html');
// Callback-Funktion zur Ausgabe der Anzahl der
// ausgewählten Benutzer-Files
function openFilesCallback(files)
{ alert('User selected ' + files.length + ' files.'); }
// Desktop-Methode zum öffnen eines Auswahl-Dialoges, in dem der User seine
// lokalen Files auswählen kann, um sie in einer Webanwendung verfügbar zu
// machen. Nach der Auswahl wird die entsprechende
// Callback-Funktion aufgerufen
desktop.openFiles(openFilesCallback);
```

Das HttpRequest-Modul

Die HttpRequest-Klasse ermöglicht das Senden von HTTP Anfragen (z.B. GET, POST, HEAD, PUT, ..), sowohl von der Haupt-HTML-Seite als auch innerhalb von Workern.

Beispiel: GET-Anfrage

```
<script type="text/javascript" src="gears_init.js"></script>
<script type="text/javascript">
// HttpRequest-Interface anlegen
var request = google.gears.factory.create('beta.httprequest') ;
// Methode und URL eines Requests spezifizieren
request.open('GET', '/index.html');
// Funktion spezifizieren, die bei Statusanzeige 4 (=Complete)
// des Requests, einen Antwort-Text auf der Console ausgibt
request.onreadystatechange = function() {
if (request.readyState == 4) { console.write(request.responseText); } };
// den Request senden
request.send();
</script>
```

Das Timer-Modul

Die Timer-Klasse ermöglicht das Setzen und Reagieren von Zeitschaltern, sowohl für die Haupt-HTML-Seite als auch innerhalb von Workern.

Beispiel: Alert über einen Timer ausgeben

```
<script type="text/javascript" src="gears_init.js"></script>
<script type="text/javascript">
// Timer anlegen
var timer = google.gears.factory.create('beta.timer');
// Setzt Timeout, der nach 1000 Millisekunden eine Funktion
// aufruft, die eine Message ausgibt
timer.setTimeout(function() { alert('Hello, from the future!'); }, 1000);
</script>
```

Das Blob-Modul

Die Blob-Klasse ermöglicht das Referenzieren von Blobs bzw. Blöcken von Binärdaten (z.B. Grafikdateien, Audiodateien), in Web-Anwendungen. Blobs können so über Methoden anderer Gears-APIs verwendet werden.

Beispiel: Blob-File-Upload über HttpRequest

```
<script type="text/javascript" src="gears_init.js"></script>
<script type="text/javascript">
// Desktop-Interface anlegen
var desktop=google.gears.factory.create('beta.desktop');
// Mit einer Desktop-Callback-Funktion einen HttpRequest zum
// uploaden eines ausgewählten Files definieren
desktop.openFiles(function(files){
var http = google.gears.factory.create('beta.httprequest');
http.open('PUT', 'http://server/'+files[0].name);
http.onreadystatechange = function() {
if (http.readyState == 4) { alert(http.responseText); }};
// den HttpRequest für das Uploaden des ausgewählten Blob-Files senden
http.send(files[0].blob);}
</script>
```

3.2 Gears-Funktionsweise

Wird eine Gears-Anwendung über den Browser eines Clients online aufgerufen, sorgt Gears nach einer Erlaubnisanfrage dafür, dass die entsprechenden Anwendungs-Ressourcen vom Webserver in den lokalen Cache gedownloaded werden.
Greift der Benutzer später offline auf die Gears-Anwendung zu, sorgt Gears dafür, dass die entsprechenden Ressourcen der Anwendung aus dem LocalServer und der Database verwendet werden. Eine Internet-Verbindung ist dann nicht mehr erforderlich.

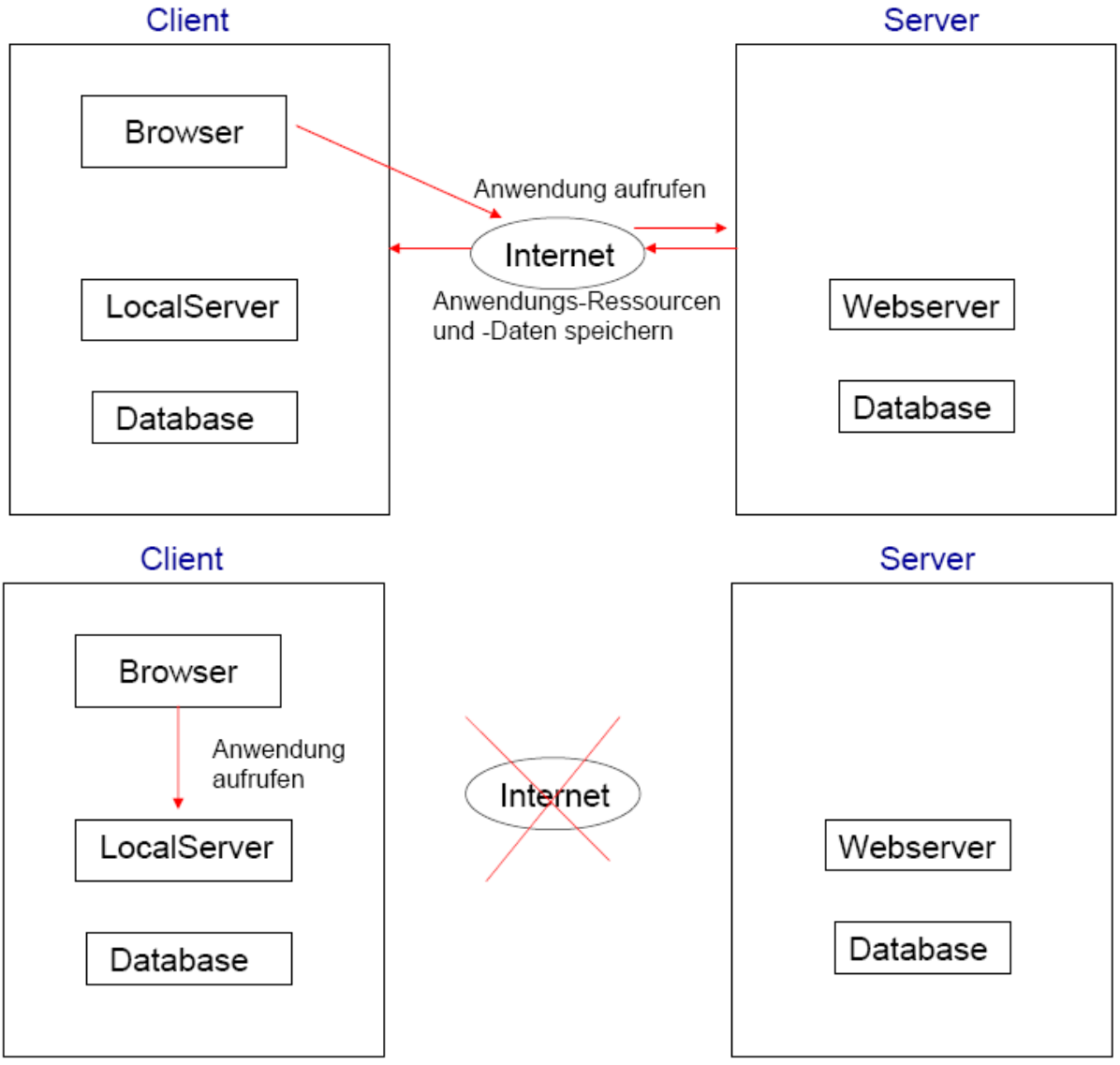


Abbildung 3: Funktionsweise von Gears

3.4 „go_offline-Website“: Eine statische Gears-Anwendung erstellen

Um zu zeigen, wie genau die Programmierung einer Gearsfähigen-Webanwendung funktioniert, im Folgenden ein erstes statisches Beispiel für eine Gearsfähige-Website.

Zu den Grundvoraussetzungen, die Entwickler für die Umgestaltung ihrer Webanwendungen zu Gearsfähigen-Webanwendungen dabei mitbringen sollten, ist anzumerken, dass die Fähigkeit JavaScript-Code zu schreiben bzw. die Basics zu verstehen von Nöten ist. Ebenfalls muß natürlich auch die Möglichkeit gegeben sein, Files über einen Http-Server zu veröffentlichen und dem Entwickler müssen alle für die offline-Anwendung benötigten Files vorliegen.

Schritt 1: Gears installieren

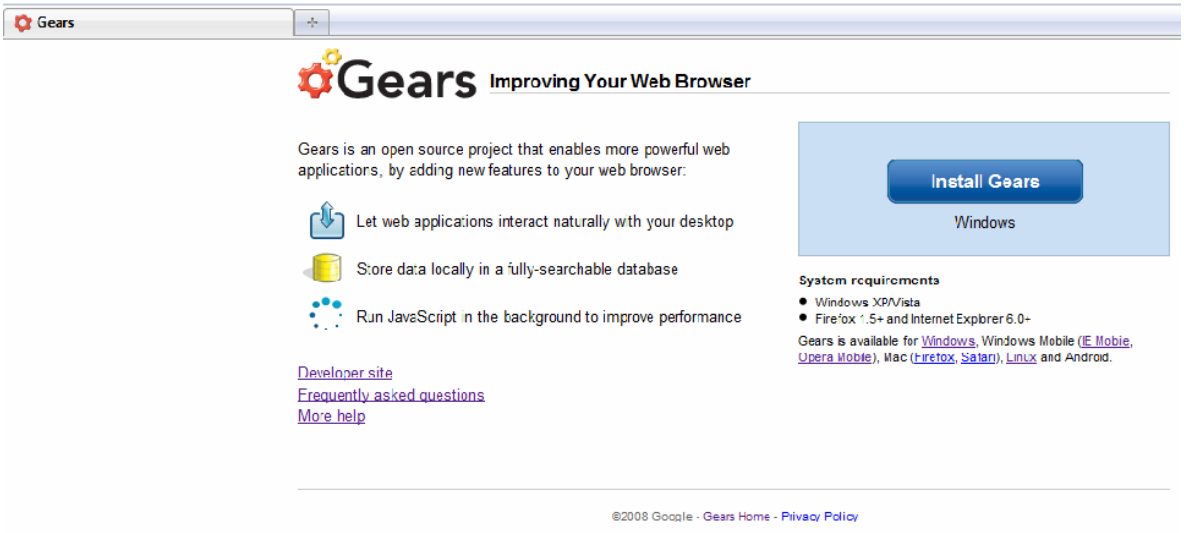


Abbildung 4: Gears-Homepage

Der erste Schritt in der Gears-Entwicklung besteht aus dem Herunterladen und Installieren des Gears-Browser-Plugins von der Gears-Homepage: `http://gears.google.com`.
Wie in Abbildung 4 zu erkennen wird beim Besuch der Website schon automatisch das für den Besucher passende Gears-Plugin für seine Plattform angeboten.

Schritt 2: Manifest-File erstellen

```
{  "betaManifestVersion": 1,
    "version": "my_version_string",
    "entries":
            [       { "url": "go_offline.html"},
                    { "url": "go_offline.js"},
                    { "url": "../gears_init.js"}     ]
}
```
Quellcode-Abbildung 1: Manifest-File „tutorial_manifest.json“

Im zweiten Schritt der Entwicklung muß ein Manifest-File erstellt werden, das die URLs der offline zu cachenden Dateien enthält (siehe Quellcode-Abb. 1).
Im Beispiel nennt sich das Manifest-File: „tutorial_manifest.json“.
„.json“ steht hierbei als Endung für „JavaScript Object Notation“.
Das ist ein unabhängiges Format für den Datenaustausch zwischen Anwendungen und kann theoretisch in jeder Programmiersprache (nicht nur in JavaScript) eingesetzt werden.

„betaManifestVersion“ gibt die Version des Manifest-File-Formats an. (1 oder 2)
„version“ nimmt eine beliebige Zeichenkette auf und liefert die Version der Datensammlung. Im Beispiel nennt sie sich nur „my_version_string“, sie könnte jedoch zum Beispiel auch Version 1 heißen. Die Versionsbezeichnung ist insofern wichtig, als dass bei unterschiedlichen Versionsbezeichnungen zwischen Manifest-File auf dem Client und Manifest-File auf dem Server, Gears automatisch die in der Server-Version verzeichneten Dateien auf das lokale System herunterlädt, um so für synchronen Datenbestand zu sorgen.
Unter „entries“ befinden sich die Dateien, die offline zur Verfügung gestellt werden sollen.
„**go_offline.html**“ ist eine Website, die im Beispiel gleichzeitig als User-Interface fungiert.
„**go_offline.js**“ ist das JavaScript-File mit den benötigten Funktionen der Website.
„**gears_init.js**“ ist ein JavaScript-File, das immer enthalten sein muß, da es zur Gears-Distribution gehört und für die korrekte Initialisierung von Gears sorgt.
Im folgenden die Browser-Ansichten von „go_offline.html“:

Handelt es sich beim Besucher der Site um jemanden, der Gears nicht installiert hat (siehe Abb. 5), erscheint in grüner Schrift die Nachricht, dass zuerst Gears installiert werden muß, was unter dem lila-Link „Install Gears“ ermöglicht wird.

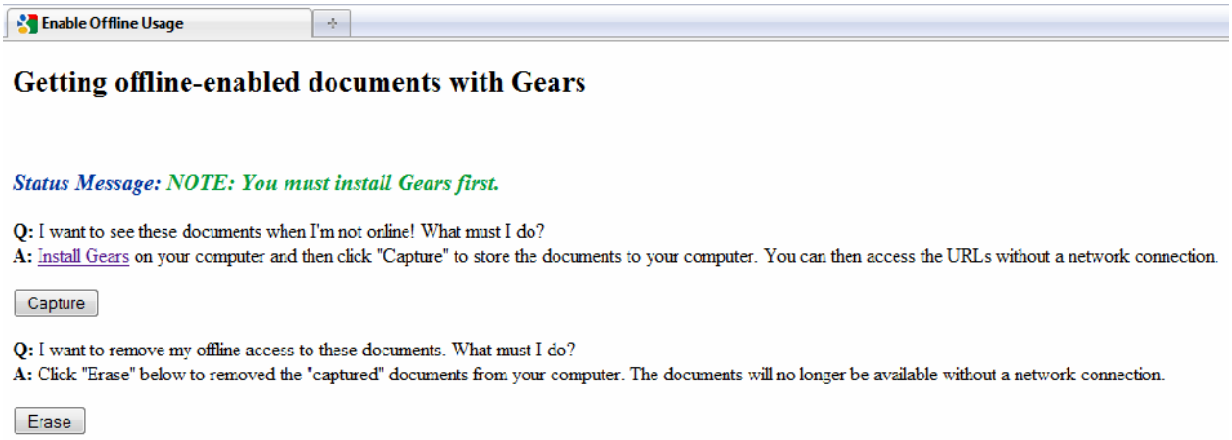


Abbildung 5: Browseransicht von „index.html“ ohne Gears-Installation

Wurde Gears installiert (siehe Abb. 6) erscheint die Nachricht, dass Gears installiert wurde und der Besucher hat die Möglichkeit durch Klick auf den Button „Capture“ die zur offline-Darstellung der Site benötigten Dateien lokal zu speichern.
Sollen die Dateien später wieder gelöscht werden, so geht das über den Button „Erase“.

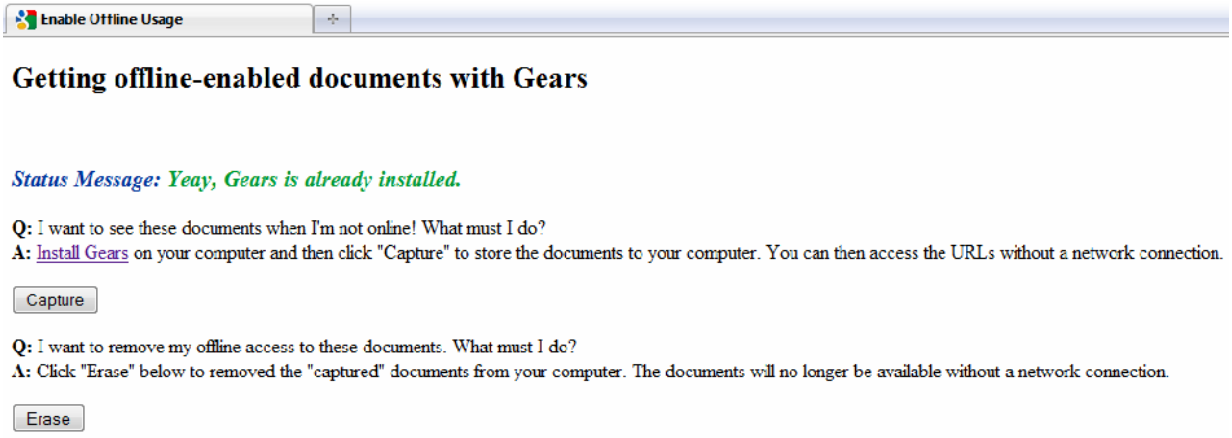


Abbildung 6: Browseransicht von „index.html“ mit Gears-installation

Schritt 3: JavaScript-Funktionen für die Nutzung der Gears-APIs einbinden

Damit die im Beispiel angebotenen Funktionalitäten zum Cachen und Löschen der Offline-Anwendungs-Ressourcen innerhalb der Webseite verwendet werden können, müssen entsprechende JavaScript-Funktionen auf der HTML-Seite „index.html“ des Beispiels eingebunden werden (in Quellcode-Abbildung 2 rot hervorgehoben).

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<script type="text/javascript" src="../gears_init.js"></script>
<script type="text/javascript" src="go_offline.js"></script>
<title>Enable Offline Usage</title>
<style type="text/css">
<!-- .style1 { color: #003399;font-style: italic;font-weight: bold;font-
size: large; }
       .style3 {color: #009933;font-weight: bold;font-style: italic; }-->
</style>
</head>
<body onload="init()">
<h2>Getting offline-enabled documents with Gears </h2>
<p> </p><div><p class="style1">
Status Message: <span id="textOut" class="style3"></span> </p></div>
<p><strong>
Q:</strong> I want to see these documents when I'm not online! What must I
do? <br />
<strong>A:</strong> <a href="http://gears.google.com/">Install Gears</a> on
your computer and then click &quot;Capture&quot; to store the documents to
your computer. You can then access the URLs without a network connection.
</p>
<p> <button onclick="createStore()" > Capture </button> </p><p><strong>
Q:</strong> I want to remove my offline access to these documents. What
must I do?
<br /><strong>A: </strong>Click &quot;Erase&quot; below to removed the
&quot;captured&quot; documents from your computer. The documents will no
longer be available without a network connection. </p>
<p> <button onclick="removeStore()" > Erase </button> </p>
</body>
</html>
```

Quellcode-Abbildung 2: Webseite index.html

„go_offline.js“ stellt das JavaScript dar, das die in „index.html“ zu implementierenden Funktionen definiert. „gears_init.js“ sorgt für die korrekte Initialisierung von Gears im Browser und gehört zur vordefinierten Gears-Distribution, muß also nicht vom Entwickler selbst geschrieben werden (für den Quellcode siehe Anhang 1).

Ein Span-Element „textOut“dient zur Ausgabe der Status-Meldung in grüner Schrift.
Die zu übergebenden Meldungen von textOut sind in go_offline.js definiert.
„createStore()“ und „removeStore()“ sind die JavaScript-Funktionen für das Erstellen und Löschen eines ManagedResourceStores für die Anwendungs-Ressourcen.
Sie werden ebenfalls in „go_offline.js“ definiert.

Schritt 4: JavaScript-Funktionen für die Nutzung der Gears-APIs definieren

Damit die im Beispiel angebotenen Funktionalitäten zum Cachen und Löschen der Offline-Anwendungs-Ressourcen, innerhalb der Webseite verwendet werden können, müssen entsprechende JavaScript-Funktionen in „go_offline.js“ überhaupt definiert werden (in der Quellcode-Abbildung 3 rot hervorgehoben).

```
var STORE_NAME = "my_offline_docset";
var MANIFEST_FILENAME = "tutorial_manifest.json";
var localServer;
var store;

function init() {
if (!window.google || !google.gears)
{    textOut("NOTE:  You must install Gears first.");  }
else {    localServer = google.gears.factory.create("beta.localserver");
             store = localServer.createManagedStore(STORE_NAME);
             textOut("Yeay, Gears is already installed.");  }}

function createStore() {
if (!window.google || !google.gears)
{    alert("You must install Gears first.");    return;  }
store.manifestUrl = MANIFEST_FILENAME;
store.checkForUpdate();
var timerId = window.setInterval(function() {
if (store.currentVersion)
{   window.clearInterval(timerId);
textOut("The documents are now available offline.\n" + "With your browser
offline, load  the document at " + "its normal online URL to see the
locally stored " +      "version. The version stored is: " +
store.currentVersion);  }
else if (store.updateStatus == 3){ textOut("Error: " +
store.lastErrorMessage); }}, 500);  }

function removeStore() {
if (!window.google || !google.gears)
{    alert("You must install Gears first.");    return;  }
localServer.removeManagedStore(STORE_NAME);
textOut("Done. The local store has been removed." + "You will now see
online versions of the documents.");}

function textOut(s) { var elm = document.getElementById("textOut");
while (elm.firstChild) { elm.removeChild(elm.firstChild); }
elm.appendChild(document.createTextNode(s));}
```

Quellcode-Abbildung 3: JavaScript go_offline.js

Als erstes wird ein Store-Name für den anzulegenden ManagedResourceStore in einer Variablen festgehalten.
Ebenso wird dies für den Manifest-Namen gemacht, bloß dass hier der schon feststehende Name des Manifest-Files „tutorial_manifest.json“ angeben wird.
Für den LokalServer als auch für den ManagedResourceStore werden Variablen deklariert, für die unter „function init()“ ein LokalServer und ein ManagedResourceStore angelegt wird.

Die Funktion „**init()**“ dient der Feststellung, ob Gears überhaupt auf dem System des Besuchers installiert ist. Ist dies nicht der Fall, wird über eine Funktion „textOut()“ die Nachricht „you must install Gears first“ an das Span-ID-Element auf der HTML-Seite übergeben.
Ist Gears allerdings installiert, werden über das Factory-API der LokalServer und über das LokalServer-API der ManagedResourceStore erstellt und an die Variablen lokalServer und store übergeben. Zudem wird über textOut die Nachricht "Yeay, Gears is already installed." übergeben.

Die Funktion „**createStore()**“ ist im HTML-Dokument mit dem „Capture“-Button implementiert, um die Ressourcen, die im Manifest-File gelistet sind auf dem lokalen Datenträger zu speichern.
Hier wird als erstes wieder überprüft, ob Gears überhaupt installiert ist, wenn nicht dann erfolgt eine Fehlermeldung.
An der Stelle „store.manifestUrl“ wird das entsprechende Manifest dem ManagedResourceStore zugewiesen und über „store.checkForUpdate()“ ein Update des ManagedResourceStores gestartet.
Sind alle im Manifest deklarierten Dateien in den ManagedResourceStore runtergeladen worden, besitzt dieser eine CurrentVersion.
Über eine if-Bedingung „store.currentversion“ wird überprüft, ob eine CurrentVersion vorliegt, ist dies der Fall wird dem Besucher der Text „The documents are now available offline. With your browser offline, load the document at its normal online URL to see the locally stored version. The version stored is: my_version_string” ausgegeben.
War das Update des ManagedResourceStore mit den im Manifest-File geforderten Dateien nicht erfolgreich, also store.updateStatus entspricht 3, was laut der Dokumentation einem fehlerhaften Update entspricht,
wird über die textout-Funktion die Fehlermeldung für den Besucher angezeigt.

Die Funktion „**removeStore()**“ ist im HTML-Dokument mit dem „Erase“-Button implementiert, und dient dem Löschen, der lokal gespeicherten Anwendungs-Ressourcen.
Hier wird wie bei createStore() wieder erst geprüft, ob Gears installiert ist, wenn nicht, erscheint wieder die Meldung, das Gears erst installiert werden muß.
Ist Gears installiert, wird der angelegte ManagedResourceStore vom LokalServer gelöscht und es erscheint die Nachricht, dass der Besucher die Inhalte der Site nicht mehr offline verwenden kann.

Die Funktion „**textOut()**“ bekommt immer einen Textstring übergeben, der dann im HTML-Dokument an der Stelle der Span-id „textout“ eingesetzt wird. Es wird hierbei immer der Textstring, der vorher dort schon gestanden hat durch den Neuen ersetzt.

Schritt 5: Files auf den HTTP-Server uploaden

Der 5te und letzte Schritt, um die Gears-fähige Website fertig zu stellen, besteht aus dem Hochladen der Files auf den Online-Server.
Konkret sind das zum einen das Manifest-File „tutorial_manifest.json“ und alle in diesem Manifest-File gelisteten Files: „go_offline.html“, „go_offline.js“, „gears_init.js“.

Test 1: Die Files lokal speichern

Nachdem die Gears-fähige Website erstellt wurde, kann die Funktionstüchtigkeit getestet werden, indem das Ablaufszenario für den Benutzer durchgespielt wird:
Zuerst wird, falls noch nicht geschehen, das Gears-Plugin installiert.
Dann wird die URL der Site „go_offline.html“ im Browser eingegeben
(zum live ausprobieren der Beispiel-Gears-Anwendung:
`http://code.google.com/intl/de-DE/apis/gears/tutorials/go_offline.html`)
Auf der erscheinenden Webseite (siehe Abb. 7) wird dann auf den Button „Capture“ geklickt, um die offline-benötigten Files aus dem Manifest-File, in einem lokalen ManagedResourceStore zu sichern.

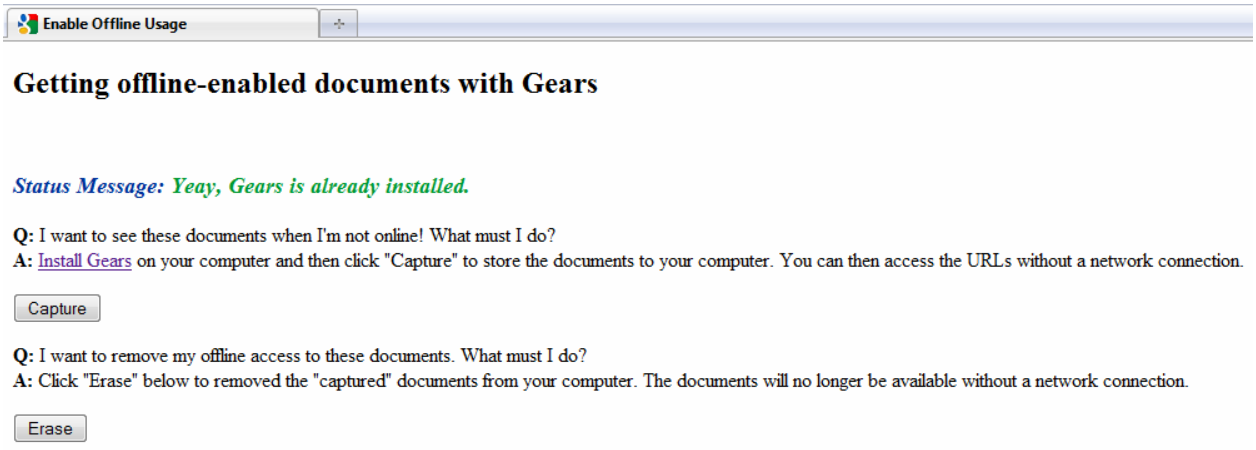


Abbildung 7: Webseite index.html vor dem Cachen

Wurde auf „Capture“ geklickt (siehe Abb. 8), erscheint in grüner Schrift die Status-Mitteilung, dass die Dokumente der Site nun offline verwendbar sind und die Website offline, über die gleiche URL wie online, im Browser abrufbar ist.

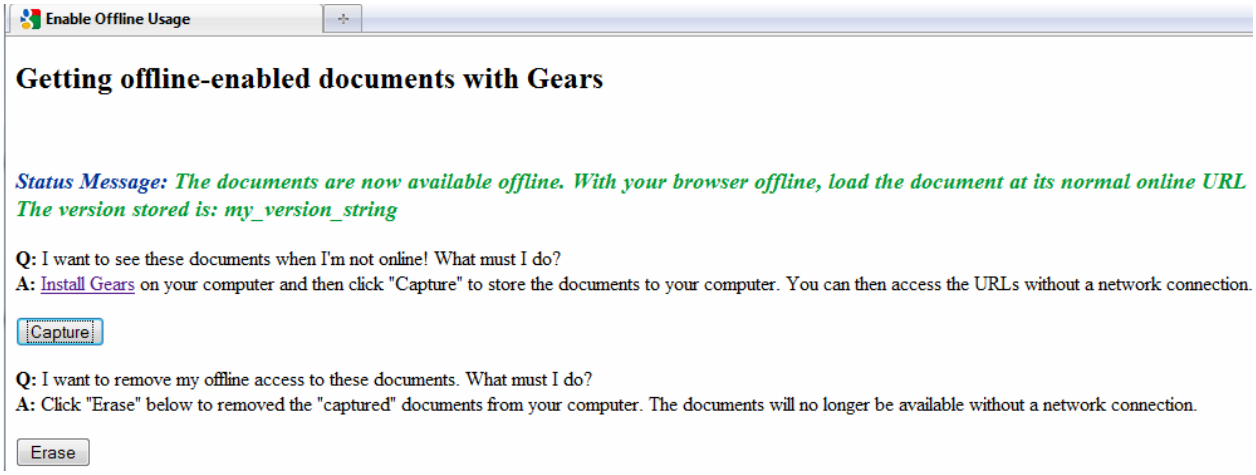


Abbildung 8: Webseite index.html nach dem Cachen

Test 2: Zugang zu den gespeicherten Files

Um nach dem lokalen Cachen nun auch die Offline-Verwendung der Gears-Website zu testen, muß der Browser zunächst offline gesetzt werden. (Bei Firefox: Datei → offline arbeiten) Danach muß der Browser-Cache geleert werden, um auch sicher sein zu können, dass wirklich das Gears-Feature zum tragen kommt. (Bei Firefox: Extras → Private Daten löschen) Die Site sollte nun auch offline über dieselbe URL abgerufen werden können.

Anmerkung: Update des Manifest-Files

Wann immer die Inhalte statischer als auch dynamischer Gears-Anwendungen aktualisiert oder Files hinzugefügt oder entfernt werden, muss auch das Manifest-File geupdatet werden. Dazu muß der Versions-String im Manifest-File geändert werden (siehe Quellcode-Abb. 4).

Erfasst Gears nämlich einen neuen Versions-String im Manifest, werden die geupdateten Inhalte automatisch auf den lokalen Datenträger des Users kopiert, wenn dieser das nächste Mal online die Anwendung besucht.

```
{  "betaManifestVersion": 1,
   "version": "my_version_string",          → z.B. “my_version_string_2“
   "entries":
              [      { "url": "go_offline.html"},
                     { "url": "go_offline.js"},
                     { "url": "../gears_init.js"}
              ]
}
```

Quellcode-Abbildung 4: Mögliche Namensänderung des Manifest-Files

4. Gears auf Mobilgeräten

Im folgenden Abschnitt werden einige wichtige Stichpunkte zu Gears auf Mobilgeräten benannt und anhand einer Schritt-Für-Schritt-Anleitung die Entwicklung einer dynamischen Gears-Anwendung für Android-Geräte gezeigt.

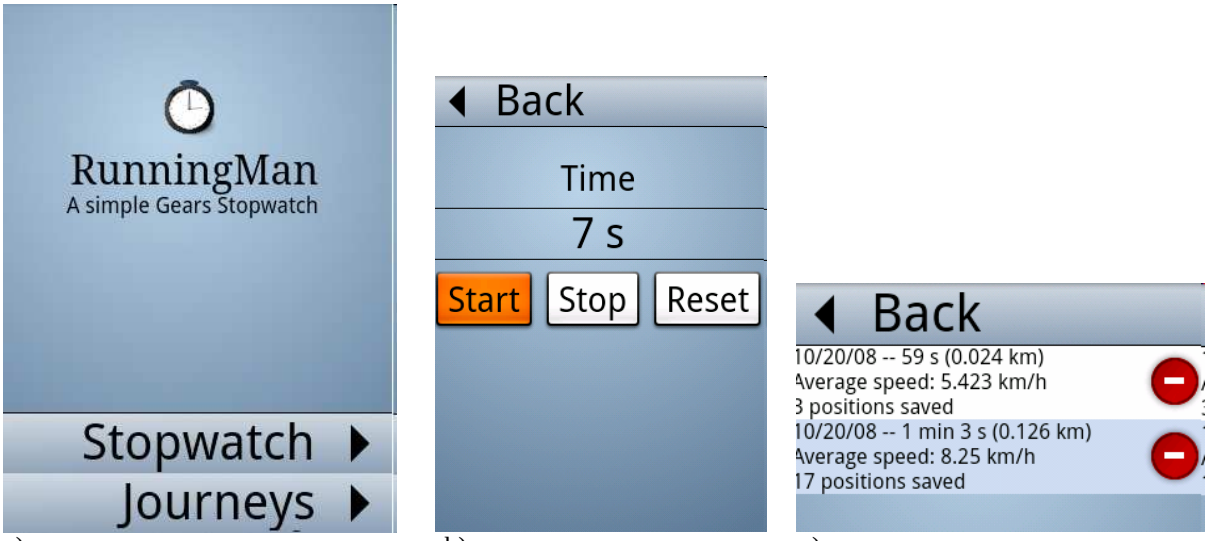
4.1 Allgemeines

Gears arbeitet exakt auf die Gleiche Weise auf Mobilgeräten wie auf PCs.
Jede Gears-Anwendung, die für den PC geschrieben wurde wird auch auf einem kompatiblen Mobilgerät laufen.
Gears ist verfügbar für Mobilgeräte mit Windows Mobile 5 oder 6 oder Android.
Sowohl für Mobilgeräte als auch für PCs besitzt Gears die gleichen APIs und folglich auch die gleichen angebotenen Funktionalitäten.
Einschränkungen beziehen sich daher lediglich auf die kleinere Bildschirmgröße und die limitierte Texteingabe, die bei der Entwicklung für Anwendungen auf Mobilgeräten ohnehin beachtet werden müssen.
Im Bezug auf den Internet Explorer Mobile auf Windows Mobile 5&6-Geräten, gibt es zudem noch Einschränkungen bei der Implementierung von CSS, DOM und ActiveX.
Einzelheiten hierzu sind unter folgendem Link einzusehen:
`http://code.google.com/intl/de-DE/apis/gears/mobile.html`

4.2 „RunningMan“: Eine dynamische Gears-Anwendung (für Android) erstellen

Da sich die Entwicklung von Gears-Anwendungen für PC und Mobilgerät ähneln, unterscheidet sich das zweite Beispiel, das hier für eine Gears-Anwendung aufzeigt wird, vom vorherigen Beispiel vor allem dadurch, dass es sich diesmal um eine dynamische Gears-Anwendung handelt und nicht um eine statische Gears-Website.
Die Gears-Anwendung wird auf Android lauffähig sein und den Namen „RunningMan“ tragen. Die Entwicklung der Gears-Anwendung wird dabei Schritt-für-Schritt erläutert, da der Quellcode etwas umfangreicher ist, als bei dem Website-Beispiel, und so der gesamte Entwicklungsprozess besser nachvollzogen werden kann.

Bei der Anwendung „RunningMan“ handelt sich um eine JavaScript-Reise-Stopuhr-Applikation, die ortsbestimmend sein soll und die Dauer, Länge und Geschwindigkeit einer Reisestrecke mittels einer Stopuhr messen und die Route anschließend auf einer Karte abbilden können soll.
Das Hauptmenü der Anwendung soll zum einen die Stopuhr „Stopwatch“ und zum anderen die Anzeige der Reiseinformationen unter „Journeys“ beinhalten (siehe Abb. 9a).



a) b) c)

Abbildung 9: „RunningMan“-Hauptmenü (a), Stopuhr (b) und Journeys-Anzeige (c)

Über „Stopwatch“ gelangt man zur Stopuhr (Abb. 9b), welche Funktionen zum Starten, Stoppen und Zurücksetzen und einen Back-Link der wieder ins Hauptmenü führt, anbieten soll.
Über „Journeys“ gelangt man zur Reisestreckenanzeige, die zu allen mit der Stopuhr getätigten Reisestrecken Informationen anzeigen soll (Abb. 9c).
Über einen Klick auf eine dieser Reisestrecken soll dann die Anzeige der zurückgelegten Strecke über eine Google-Maps-Karte erfolgen (siehe Abb. 10).
Diese Karte soll über Plus und Minus zoombar und im Anzeigebereich nach links, rechts, oben und unten verschiebbar sein.

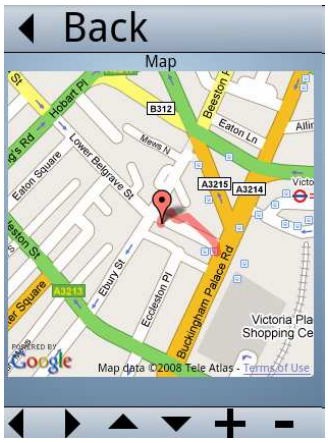


Abbildung 10: Google-Maps-Karte

Die Gears-APIs, die für die Anwendung zu nutzen sind, umfassen:

- das **LocalServer**-API für das Speichern der Anwendungs-Ressourcen,
- das **Database**-API, um Einstellungen, Laufzeiten und Routen-Positionen zu speichern,
- das **Geolocation**-API, über das die Standortbestimmung ermöglicht wird und
- das **Desktop**-API, das eine Anwendungs-Verknüpfung auf dem Desktop eines Android-Handys anlegen soll.

Zusätzlich zu diesen Gears-APIs wird noch das **Google-Maps**-API benötigt.
Für die Verwendung des Google-Maps-API ist ein Google-Maps-API-Key von Nöten.
Dieser ist erst nach der Anmeldung eines kostenlosen Google Accounts unter folgendem Link erhältlich: `http://code.google.com/apis/maps/signup.html`.

Schritt 1: Die Basis-Applikation

Der erste Schritt besteht aus der Programmierung der Basis der RunningMan-Anwendung.
Neben einem CSS-File „run.css“ und einigen Bildern, besteht die Basis grundlegend aus einem HTML-File „**index.html**“ für die Darstellung und einem JavaScript-File „**model.js**“, das die in index.html zu verwendenden Funktionen definiert.

```
<html><head><title>RunningMan</title>
<link rel="stylesheet" type="text/css" href="../run.css" />
<script type="text/JavaScript" src="model.js"></script>
<meta name="viewport" content="width=320; user-scalable=false">
</head><body>
<div id="mainScreen" align="center">
<div id="title"><img src="../images/icon.png" width="48" height="48">
<br>RunningMan<div id="subtitle">A simple Gears Stopwatch</div>
</div><div class="main-menu" align="right">
<div class="go-button" onclick="go('watch')">Stopwatch</div>
<div class="go-button" onclick="go('journeys')">Journeys</div>
</div></div>
<div id="watch" align="center">
<div class="content">Here goes the stopwatch</div>
<div class="menu" align="left">
<div class="back-button" onclick="go('mainScreen')">Back</div>
</div></div>
<div id="journeys" >
<div class="content">Here goes the journey</div>
<div class="menu" align="left">
<div class="back-button" onclick="go('mainScreen')">Back</div></div></div>
</body></html>
```

Quellcode-Abbildung 5: RunningMan - index.html

Die HTML-Seite „index.html“ (siehe Quellcode-Abb. 5) ist in mehrere Bereiche unterteilt, von denen immer nur einer zur Zeit im Fenster der Anwendung dargestellt wird.
Diese Bereiche sind in Quellcode-Abbildung 5 in blau dargestellt.
Sie umfassen zum einen den Bereich „mainScreen“, für die Darstellung der Hauptmenü-Seite, den Bereich „watch“, für die Darstellung der Stopuhr-Seite, und den Bereich „journeys“ für die Darstellung der Reiseroutenseite.
Die 3 Bereiche werden jeweils über den Aufruf einer go-Funktion (in Quellcode-Abbildung 5 in rot dargestellt) beim Klicken auf einen Hauptmenü- oder Back-Link, angezeigt oder versteckt.

Das JavaScript-File „model.js“ (siehe Quellcode-Abb. 6) definiert die Funktion „go()“ wie noch 3 andere für go() wichtige Hilffunktionen showDiv(), hideDiv() und hideAllScreens(). „showDiv()“ stellt jeweils nur einen bestimmten der 3 Bereiche des HTML-Codes dar, „hideDiv()“ macht genau das Gegenteil, versteckt also einen bestimmten der 3 Bereiche und hideAllScreens() versteckt alle 3 Bereiche.
Die Funktion „go()“ funktioniert nun so, dass sie mit hideAllScreens() erst alle 3 bereiche versteckt und dann über showDiv() einen speziellen Bereich anzeigt.

```
function go(name) {
  hideAllScreens();
  var functionName = "on_" + name;
  showDiv(name);
  if (window[functionName] != null) {
    window[functionName](); }}

function showDiv(name) {
  var elem = document.getElementById(name);
  if (elem) {
    elem.style.display="block"; }}

function hideDiv(name) {
  var elem = document.getElementById(name);
  if (elem) {
    elem.style.display="none"; }}

function hideAllScreens() {
  hideDiv('mainScreen');
  hideDiv('watch');
  hideDiv('journeys');}
```

Quellcode-Abbildung 6: RunningMan – model.js

Schritt 2: Die Stopuhr

Für die Stopuhr muss zunächst ein globales Objekt mit Namen „global“ in „model.js“ deklariert werden: `var global = new Object();`

Damit eine Main-Methode, zum initialisieren der Stopuhr, direkt beim Start der HTML-Seite geladen werden kann, wird in „index.html“ eine Main-Methode eingebaut:
`<body onload="main()">.`

Definiert wird die Main-Methode in „model.js“:
`function main() { global.startTime = null; updateTime(); }`

Sie setzt die Startzeit der Stopuhr auf null und veranlasst über eine Funktion „updateTime()“ die Ausgabe der verstrichenen Zeit.

Damit der User die Stopuhr überhaupt starten, stoppen und zurücksetzen kann, müssen als nächstes die entsprechenden Buttons und ihre verknüpften Funktionen implementiert werden, was durch modifizieren des „watch“-Bereiches in index.html geschieht. Dort werden die 3 Funktionen startRun(), stopRun() und resetRun() eingebunden (siehe Quellcode-Abb. 7).

```
<div id="watch" align="center"><div class="content">
<div id="timeTitle">Time</div>
<div id="timeDisplay"></div>
<input type="button" onclick="startRun()" value="Start"></input>
<input type="button" onclick="stopRun()" value="Stop"></input>
<input type="button" onclick="resetRun()" value="Reset"></input>
</div><div class="menu" align="left">
<div class="back-button" onclick="go('mainScreen')">Back</div>
</div></div>
```

Quellcode-Abbildung 7: Einbinden der Stopuhr-Funktionen in index.html

Somit müssen insgesamt 4 weitere Funktionen definiert werden:
Zum einen die 3 Stopuhr-Funktionen startRun(), stopRun() und resetRun() und die Funktion updateTime() der Main-Methode. Dies erfolgt wieder im JavaScript „model.js“
(siehe Quellcode-Abb. 8)

```
function startRun() {
global.updateInterval = setInterval("updateTime()", 1000);
global.startTime = new Date(); }

function stopRun() { clearInterval(global.updateInterval); }

function resetRun() { stopRun(); global.startTime = null; updateTime(); }

function updateTime() {
var time = 0;
if (global.startTime != null) {
                        time = (new Date()).getTime() - global.startTime;
                        time = (time/1000)|0; }
var timeDiv = document.getElementById("timeDisplay");
timeDiv.innerHTML = formatTime(time); }

function formatTime(aTime) {
var seconds = aTime % 60;
var minutes = ((aTime / 60) |0) % 60;
var hours = (aTime / 3600) |0;
var time = "0";
if (seconds > 0) {  time = seconds + " s";}
if (minutes > 0) {  time = minutes + " min " + time;}
if (hours > 0) {  time = hours + " h " + time;}
return time; }
```

Quellcode-Abbildung 8: Definieren der Stopuhr-Funktionen und der Anzeige-Funktionen

Die Funktion „startRun()“ speichert die aktuelle Zeit im „global“-Objekt „global.starttime“ und aktualisiert die verstrichene Zeit jede Sekunde über die „updateTime()“-Funktion.
Die Funktion „stopRun()“ leert das Timer-Objekt, so dass „updateTime()“ nicht mehr ausgeführt wird und sich die Anzeige nicht mehr ändert, also stehen bleibt.
Die Funktion „resetRun()“ hält die Stopuhr über „stopRun()“ an, setzt „global.starttime“ wieder auf null und aktualisiert dann die Anzeige über „updateTime()“ um die Zurücksetzung anzuzeigen.
Die Funktion „updateTime()“ berechnet die verstrichene Zeit als Differenz von aktueller Zeit minus Startzeit und verwendet für die Anzeige der verstrichenen Zeit eine Hilfsfunktion „formatTime()“, welche die verstrichene Zeit in Sekunden, Minuten und Stunden umrechnet.

Schritt 3: Shortcut hinzufügen

Im nächsten Schritt folgt die erste Verwendung eines Gears-APIs, nämlich des Desktop-APIs, um ein Desktop-Icon, also eine Verknüpfung unserer Anwendung auf dem Desktop von Android, anzulegen (siehe Abb.).
Hierzu wird zunächst, da ein Gears-API verwendet werden soll, das JavaScript-File „gears_init.js“ für die Initialisierung in „index.html“ eingebunden (siehe Quellcode-Abb. 9).

```
<head>
<title>RunningMan</title>
<link rel="stylesheet" type="text/css" href="../run.css" />
<script type="text/JavaScript" src="gears_init.js"></script>
<script type="text/JavaScript" src="model.js"></script>
<meta name="viewport" content="width=320; user-scalable=false">
</head>
```

Quellcode-Abbildung 9: Einbinden von „gears_init.js“

Ist dies erledigt, wird der Pfad der „index.html“-Site noch in der Main-Methode hinzugefügt und eine Funktion „installShortcut()“ darin aufgerufen, welche für die Installation des Desktop-Icons sorgen soll und ebenfalls selbst definiert wird (siehe Quellcode-Abb. 10).

```
function main() {
global.startTime = null;
updateTime();
global.siteUrl = "http://YOUR_URL/index.html";
installShortcut(); }

function installShortcut() {
var desktop = google.gears.factory.create('beta.desktop');
desktop.createShortcut("RunningMan", global.siteUrl,
{ "48x48" : "../images/icon.png" }, "RunningMan application");}
```

Quellcode-Abbildung 10: Definieren der installShortcut()-Funktion

„installShortcut()“ erzeugt über die Gears-Factory eine Desktop-Instanz, über deren API-Methode „createShortcut()“ dann das entsprechende Icon für „RunningMan“ installiert wird. Hierfür werden ein Name für die Anwendung, die URL, die in der Main-Methode definiert wurde, ein Bild für das Icon und eine kurze Beschreibung der Anwendung übergeben.

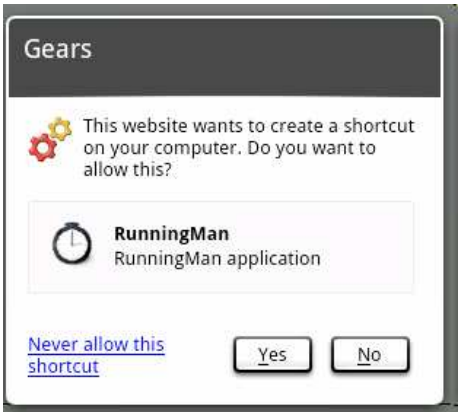


Abbildung 11: Shortcut-Installations-Fenster

Abbildung 12: Shortcut auf dem Android-Desktop

Das Szenario für den User sieht wie in Abbildung 11 und 12 zu erkennen so aus, dass wenn er die Seite index.html besucht und Gears installiert hat, er über ein Fenster gefragt wird, ob er die Erlaubnis für die Installation eines Desktop-Icons für RunningMan erteilt (Abb. 11).
Klick er auf „Yes“ wird das Icon auf dem Android-Desktop installiert (Abb. 12).
Wann immer der User auf dieses Icon klickt, öffnet sich die RunningMan-Anwendung im Browser.

Das Problem zu diesem Zeitpunkt besteht jedoch darin, dass selbst wenn das Icon schon installiert wurde, der User dennoch bei jedem öffnen der Anwendung, nochmals gefragt wird, ob er das Icon installieren möchte.
Um dies zu verhindern muss der Anwendung ermöglicht werden, Informationen bzw. Einstellungen des Users als Preferenzen speichern zu können.

Schritt 4: Preferenzen speichern

Um das Speichern von Informationen zu ermöglichen, wird eine Datenbank bzw. das Database-API von Gears benötigt.
Für das Anlegen der Datenbank wird in model.js eine neue Funktion namens „initDB()“ hinzugefügt (siehe Quellcode-Abb. 11).

```
function initDB() {
global.db = google.gears.factory.create('beta.database');
global.db.open('stopwatch');
global.db.execute('CREATE TABLE IF NOT EXISTS Preferences ' +
                  '(Name text, Value int)');}

function getPreference(name) {
var result = false;
var rs = global.db.execute('SELECT Value FROM Preferences
                            WHERE Name = (?)',   [name]);

if (rs.isValidRow()) { result = rs.field(0);}
rs.close();
return result; }

function setPreference(name, value) {
global.db.execute('INSERT INTO Preferences VALUES (?, ?)', [name, value]) }
```

Quellcode-Abbildung 11: Definition der Database mit Preferenzen-Table und der Preferenzen-Funktionen

Die Funktion „initDB()“ erstellt über das Factory-API eine Datenbank und legt eine Preferenzen-Tabelle mit den Attributen „Name“ und „Value“ an (siehe Abb. 13).

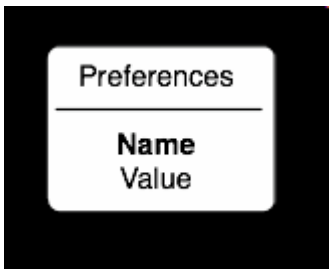


Abbildung 13: Schema der Preferenzen-Tabelle

Die Preferenzen-Tabelle wird angelegt, um zu speichern, ob ein Desktop-Icon für die Anwendung bereits installiert ist oder nicht; soll also seine Preferenz enthalten.
Um den Preferenz-Wert Abzurufen und Einzufügen dienen eine „getPreference()“- und „setPreference()“-Funktion als Hilfsfunktionen (siehe Quellcode-Abb. 11).
Die Funktion getPreference() ist für das Abrufen von Werten aus der Preferenzen-Tabelle und die Funktion setPrefernce() für das Einfügen von Werten in die Preferenzen-Tabelle zuständig. Realisiert werden die Funktionen, wie bei initDB() über SQL-Statements an die angelegte Datenbank.

Um direkt nach dem Installieren des Desktop-Icons, in der Preferenzen-Tabelle abzuspeichern, dass nun bereits ein Shortcut installiert wurde, muss der Preferenz-Wert von Shortcut in der Preferenzen-Tabelle auf true gesetzt werden.
Hierfür werden die Funktionen getPreference() und setPreference() in der Funktion installShortcut() aufgerufen. (siehe Quellcode-Abb. 12).

```
function installShortcut() {
if (getPreference('Shortcut') == false) {
var desktop = google.gears.factory.create('beta.desktop');
desktop.createShortcut("RunningMan", global.siteUrl,
{ "48x48" : "../images/icon.png" }, "RunningMan application");
setPreference('Shortcut', true);} }
```

Quellcode-Abbildung 12: Modifizieren von installShortcut()

Hat in der Prefernzen-Tabelle der Eintrag für Shortcut den Wert false, ist noch kein Desktop-Icon installiert, und die Installation wird vorgenommen.
Anschließend wird über setPreference() der Wert für Shortcut als true gespeichert, so dass beim nächsten Aufruf von RunningMan gespeichert ist, dass das Icon schon installiert wurde.

Da die Datenbank zum Speichern der Preferenzen-Werte logischerweise vor der Überprüfung für die Icon-Installation vorhanden sein muß, muß noch die Main-Methode dahingehend modifiziert werden, dass sie initDB() vor der Ausführung von installShortcut() aufruft (siehe Quellcode-Abb. 13).

```
function main() {
global.startTime = null;
updateTime();
global.siteUrl = "http://YOURURL/index.html";
initDB();
installShortcut();}
```

Quellcode-Abbildung 13: Modifizieren der main()-Methode

Die Anwendung ist nun in der Lage eine Stopuhr anzubieten und ein Desktop-Icon anzulegen.

Schritt 5: Laufzeiten speichern

Bei der Auswahl von Reiserouten spielt die Zeit eine wesentliche Rolle, weswegen die Speicherung der verschieden Laufzeiten des Nutzers, für eine spätere Darstellung, eine Sinnvolle Ergänzung der Reise-Stopuhr darstellt.

Für die Laufzeiten wird wieder, wie bei den Preferenzen geschehen, über die Funktion initDB(), die Erstellung einer Tabelle als SQL-Statement an die Datenbank ausgeführt (siehe Quellcode-Abb. 14). Die neue Tabelle nennt sich „Times“ und hat die Attribute „StartDate“, „StopDate“ und „Description“.
„StartDate“ soll das Startdatum inclusive Uhrzeit, „StopDate“ das Stopdatum incl Uhrzeit und „Description“ die Laufzeitdauer speichern.

```
function initDB() {
global.db = google.gears.factory.create('beta.database');
global.db.open('stopwatch');
global.db.execute('CREATE TABLE IF NOT EXISTS Preferences ' +
'(Name text, Value int)');
global.db.execute('CREATE TABLE IF NOT EXISTS Times ' +
'(StartDate int, StopDate int, Description text)');}
```

Quellcode-Abbildung 14: Times-Tabelle anlegen

Um die Werte für die Start- und Stop-Attribute zu erhalten und in der Tabelle einzufügen, müssen die Funktionen startRun() und stopRun() modifiziert werden (siehe Abb. 15).

```
function startRun() {
global.updateInterval = setInterval("updateTime()", 1000);
global.startTime = new Date();
var time = global.startTime.getTime();
global.db.execute('INSERT INTO Times (StartDate) VALUES (?)', [time]); }

function stopRun() {
clearInterval(global.updateInterval);
var stopDate = new Date();
var time = stopDate.getTime();
global.db.execute('UPDATE Times SET StopDate = (?) ' +
                  'WHERE ROWID = (?)', [time, global.db.lastInsertRowId]); }
```

Quellcode-Abbildung 15: startRun() und stopRun() modifizieren

Beim Ausführen der Funktion startRun() wird ein neuer Eintrag in der Tabelle Times mit dem jeweiligen Datum und Uhrzeit in der Times-Tabelle eingefügt.
Wird stopRun() ausgeführt, wird im zuletzt in die Tabelle Times angelegten Eintrag, also dem aktuellen Eintrag, der Wert für die Stopzeit eingetragen.

Die Anwendung ist nun in der Lage die Startzeit und die Stopzeit zu speichern.
Was noch fehlt ist die Speicherung von Description-Werten, also den Laufzeitdauer-Werten, wie auch die Anzeige dieser Werte auf der Journeys-Seite der Anwendung.

Schritt 6: Laufzeiten anzeigen

Da bei der ersten Ausführung der Anwendung noch keine Laufzeiten gespeichert sein werden, muß für diesen Fall die „index.html“-Seite dahingehend modifiziert werden
(siehe Quellcode-Abb. 16), dass im Bereich „Journeys“, in dem die Laufzeiten anzeigt werden sollen, ein Standardtext erscheint, falls noch keine Laufzeiten vorhanden sind (siehe Abb. 14).

```
<div id="journeys"> <div class="content">
<div id="journeysContent">
<h2>No journeys yet!</h2></div> </div>
<div class="menu" align="left">
<div class="back-button" onclick="go('mainScreen')">Back</div></div></div>
```

Quellcode-Abbildung 16: Der journeys-Bereich, wenn keine Einträge vorhanden

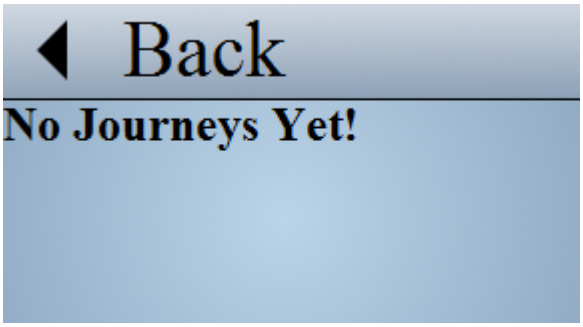


Abbildung 14: Die „journeys“-Seite ohne Laufzeit-Einträge

Wie die Laufzeiten ausgegeben werden sollen, wenn sie vorhanden sind, zeigt Abbildung 15. Um diese Anzeige zu realisieren wird eine Funktion „on_journeys()“ benötigt, die für die Speicherung und Ausgabe der Laufzeitdauer-Werte sorgen soll (siehe Quellcode-Abb. 17).

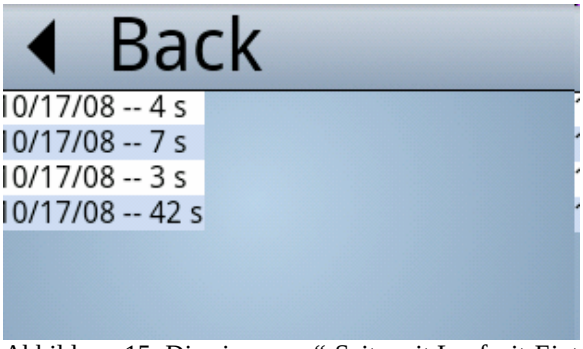


Abbildung 15: Die „journeys“-Seite mit Laufzeit-Einträgen

```
function on_journeys() {
var rows = 0;
var html = "";
var rs = global.db.execute('SELECT ROWID, Description FROM Times');

while (rs.isValidRow()) {
  var rowID = rs.field(0);
  var description = rs.field(1);
  if (description == null) { description = createDescription(rowID) ; }
  var rowType;
  if (rows % 2 == 0) { rowType = "rowA"; }
  else { rowType = "rowB"; }
  rows++;
  html += "<div class='" + rowType + "'>";
  html += "<div class='rowDesc'>" + description + "</div>";
  html += "</div>";
  rs.next(); }
  if (html != "") {  var elem = document.getElementById('journeysContent');
                     elem.innerHTML = html; }  }
```

Quellcode-Abbildung 17: Die Funktion „on_journeys()“

Die Funktion „on_journeys()“ wird immer dann aufgerufen, wenn der User im Hauptmenü „Journeys“ auswählt und gespeicherte Laufzeiten vorhanden sind.
Sie iteriert sich über eine While-Schleife durch die gespeicherten Einträge aus der Times-Tabelle und holt deren Description bzw. deren Laufzeitdauer aus der Tabelle, um sie anzuzeigen.
Ist für „description“ nichts eingetragen, wird über eine if-Anweisung eine Funktion „createDescription()“ aufgerufen, die für die Berechnung der Laufzeitdauer eines Eintrages sorgen (siehe Quellcode-Abb. 18) und diese dann in der Times-Tabelle abspeichern soll.
Die einzelnen Laufzeiteinträge werden jeweils in blau oder weiß unterlegt (siehe Abb. 15). Hierfür wird für jeden Laufzeiteintrag berechnet, ob es sich um einen Geraden oder Ungeraden handelt. Ist der Laufzeiteintrag gerade, wird er dem Typ „rowA“ zugeordnet und erscheint blau unterlegt, ist der Laufzeiteintrag ungerade, wird er dem Typ „rowB“ zugeordnet und erscheint weiß unterlegt. Die entsprechenden Formatierungseinstellungen sind innerhalb des CSS-Files „run.css“ festgelegt (siehe Anhang 2).

Die Funktion „createDescription()“, dient wie bereits erwähnt, der Berechnung und Abspeicherung von Laufzeitdauer-Werten in der Times-Tabelle.
Sie holt sich die Start- und Stopzeit eines Laufzeiteintrages, berechnet die Differenz und wendet auf diesen Wert, dann die Funktion „formatTime()“ an, um ihn in Sekunden, Minuten oder Stunden dann als description-Wert zu speichern.

```
function createDescription(rowID) {
var description = "";
var rs = global.db.execute('SELECT StartDate, StopDate FROM Times
                            ' + 'WHERE ROWID = (?)', [rowID]);
if (rs.isValidRow()) {
 var sDate = rs.field(0);
 var eDate = rs.field(1);
 var time = (((eDate - sDate)/1000)|0); // elapsed seconds
 var startDate = new Date();
 startDate.setTime(sDate);
 description = startDate.toLocaleDateString() + " -- ";
 description += formatTime(time);
 global.db.execute('UPDATE Times SET Description = (?) ' +
                   'WHERE ROWID = (?)', [description, rowID]);}
 return description;}
```

Quellcode-Abbildung 18: Die createDescription()-Funktion

Was bis dato noch sehr unpraktisch ist, ist dass die Laufzeiten zwar nun gespeichert und angezeigt werden können, jedoch nicht wieder entfernt.

Schritt 7: Laufzeiten entfernen

Um das Löschen von Laufzeiten zu realisieren muß die Funktion „on_journeys“, für die Anzeige der Laufzeiten, ein Lösch-Icon (siehe Abb. 16) mit einer entsprechenden Löschfunktion „deleteRecord()“ einbinden (siehe Quellcode-Abb. 19):

```
html += "<div class='" + rowType + "'>";
html += "<div class='rowDesc'>" + description + "</div>";
html += "<div class='rowImg' onclick='deleteRecord(" + rowID + ")'>";
html += "<img src='delete.png'></div";
html += "</div>";
```

Quellcode-Abbildung 19: Modifizieren von on_journeys

Die Funktion „deleteRecord()“ (siehe Quellcode-Abb. 20) ruft zunächst einen Frage-Text auf, der den Benutzer fragt, ob der Eintrag wirklich gelöscht werden soll.
Bestätigt der Benutzer den Löschvorgang, wird der Eintrag gelöscht und anschließend die Anzeige erneuert.

```
function deleteRecord(rowID) {
var answer = confirm("Delete this run?");
if (answer) {
global.db.execute('DELETE FROM Times WHERE ROWID = (?)',
[rowID]);
go('journeys');
}}
```

Quellcode-Abbildung 20: Die deleteRecord()-Funktion

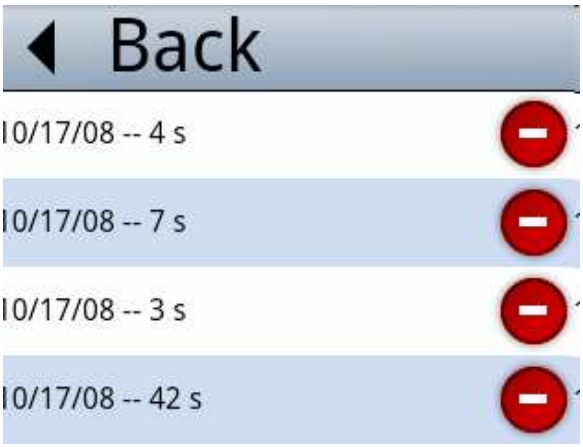


Abbildung 16: Die „journeys“-Seite mit Laufzeiten und Lösch-Icons

Schritt 8: Geolocation-Information speichern

Um für die Laufzeiten neben der Dauer auch die Distanz und Geschwindigkeit anzeigen zu können, ist die Positionsbestimmung von grundlegender Bedeutung.
In Gears wird das Geolocation-API für die Standortbestimmung verwendet.
Um die durch das API erhaltenen Informationen zu einer Position zu speichern, wird eine weitere Tabelle benötigt. Diese Positions-Tabelle wird wie auch die anderen Tabellen, mit der Datenbank über initDB() angelegt (siehe Quellcode-Abb. 21).

```
function initDB() {
global.db = google.gears.factory.create('beta.database');
global.db.open('stopwatch');
global.db.execute('CREATE TABLE IF NOT EXISTS Preferences ' +
'(Name text, Value int)');
global.db.execute('CREATE TABLE IF NOT EXISTS Times ' +
'(StartDate int, StopDate int, Description text)');
global.db.execute('CREATE TABLE IF NOT EXISTS Positions ' +
'(TimeID int, Date int, Latitude float, ' +
' Longitude float, Accuracy float, Altitude float, ' +
' AltitudeAccuracy float)');
}
```

Quellcode-Abbildung 21: Positions-Tabelle anlegen

Die Positions-Tabelle enthält Attribute zur Laufzeit und Datumsbestimmung, sowie zu Latitude - Breitengrad, Longitude - Längengrad, Accuracy - Genauigkeit, Altitude - Höhenlage und AltitudeAccuracy - Höhenlagengenauigkeit.
Das Geolocation-Objekt an sich wird wie auch die Datenbank über die Main-Methode angelegt (siehe Quellcode-Abb. 22).

```
function main() {
global.startTime = null;
global.geo = google.gears.factory.create('beta.geolocation');
updateTime();
global.siteUrl = "http://YOURURL/index.html";
initDB();
installShortcut();}
```

Quellcode-Abbildung 22: Geolocation-Objekt anlegen

Funktionieren soll die RunningMan-Anwendung mit Hilfe des Geolocation-API nun so, dass wenn die Stoppuhr startet die Position des Android-Handys gespeichert wird, dann beim gehen noch weitere Zwischenstationen gespeichert werden und beim stoppen der Uhr ebenfalls noch mal die Position gespeichert wird.
Um diese Speicherungen zu realisieren müssen die startRun()- und stopRun()-Funktion modifiziert werden.

```
function startRun() {
global.updateInterval = setInterval("updateTime()", 1000);
global.startTime = new Date();
var time = global.startTime.getTime();
global.db.execute('INSERT INTO Times (StartDate) VALUES (?)', [time]);
global.currentTimeID = global.db.lastInsertRowId;
global.currentGeoWatcher = global.geo.watchPosition(
function(position) {
global.db.execute('INSERT INTO Positions (TimeID, Date,Latitude, ' +
                  'Longitude, Accuracy, Altitude, AltitudeAccuracy) ' +
                  'VALUES (?, ?, ?, ?, ?, ?, ?)',
                  [global.currentTimeID, position.timestamp.getTime(),
                  position.latitude, position.longitude,
                  position.accuracy,
                  position.altitude, position.altitudeAccuracy]);
                  },
null, { "enableHighAccuracy" : true});
}
```

Quellcode Abbildung 23: startRun() modifizieren

Die Funktion startRun() wird dahingehend modifiziert, dass beim starten der Stopuhr, nicht mehr nur die Startzeit gespeichert wird, sondern zusätzlich noch ein Geolocation-Watcher gestartet wird (siehe Quellcode-Abb. 23).
Dieser erhält eine Benachrichtigung sobald sich die aktuelle Position ändert und speichert dann über eine Funktion „funktion(position)“ die neue Position in der Positions-Tabelle ab.
Über das Argument „enableHighAccuracy: true“ wird zudem die beste verfügbare Funktion für die Positionsbestimmung genutzt, welche auf Android g1-Geräten der Positionsbestimmung über GPS entspricht.

```
function stopRun() {
clearInterval(global.updateInterval);
var stopDate = new Date();
var time = stopDate.getTime();
global.db.execute('UPDATE Times SET StopDate = (?) ' +
'WHERE ROWID = (?)', [time, global.currentTimeID]);
if (global.currentGeoWatcher != null) {
global.geo.clearWatch(global.currentGeoWatcher);
global.currentGeoWatcher = null;
}}
```
Quellcode-Abbildung 24: stopRun() modifizieren

Wird die Stopuhr gestoppt, muss der Geolocation-Watcher ebenfalls gestoppt werden. Das Stoppen erfolgt über die Funktion stopRun(), indem der Geolocation-Watcher gleich null gesetzt wird (siehe Quellcode-Abb. 24).

Damit beim löschen von Laufzeiteinträgen auch die damit verbundenen Positionsdaten gelöscht werden, muss zudem noch die Funktion deleteRecord() entsprechend modifiziert werden (siehe Quellcode-Abb. 25).

```
function deleteRecord(rowID) {
var answer = confirm("Delete this run?");
if (answer) {
global.db.execute('DELETE FROM Positions WHERE TimeID = (?)',
[rowID]);
global.db.execute('DELETE FROM Times WHERE ROWID = (?)',
[rowID]);
go('journeys');
}}
```
Quellcode-Abbildung 25: deleteRecord() modifizieren

Da nun die Positionsdaten gespeichert werden können, erfolgen im nächsten Schritt die Berechnungen für Distanz und Geschwindigkeit, auf Grundlage der Positionsdaten, und die entsprechende Ausgabe der Werte mit den Laufzeiteinträgen.

Schritt 9: Modifizieren der Übersicht

Um die Informationen über Distanz und Geschwindigkeit zusammen mit den Laufzeiten auszugeben (siehe Abb. 17), werden diese Informationen über eine Funktion „positionInformation()“ - innerhalb der Funktion createDescription(), die ja für die Laufzeitdauer-Einträge in der Tabelle Times gedacht ist - angehängt:

```
description = startDate.toLocaleDateString() + " -- ";
description += formatTime(time); description += positionInformation(rowID);
```

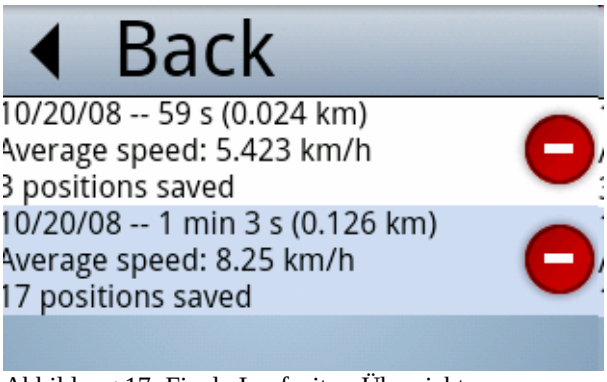


Abbildung 17: Finale Laufzeiten-Übersicht

```
function positionInformation(rowID) {
var distance = 0;
var prevLat = null;
var prevLon = null;
var firstTime = 0;
var lastTime = 0;
var nbPositions = 0;
var rs = global.db.execute('SELECT Latitude, Longitude, Date ' +
                           'FROM Positions WHERE TimeID = (?) ' +
                           'ORDER BY Date', [rowID]);
         while (rs.isValidRow()) {
              nbPositions++;
              var lat = rs.field(0);
              var lon = rs.field(1);
              var date = rs.field(2);
              if (firstTime != 0) {
              distance += haversineDistance(prevLat, prevLon, lat, lon) ; }
              else { firstTime = date; }
              prevLat = lat;
              prevLon = lon;
              lastTime = date;   rs.next(); }
var secTime = (lastTime - firstTime) / 1000;
var averageSpeed = ((distance * 3600) / secTime);
var roundedDistance = (((distance*1000)|0)/1000);
var roundedSpeed = ((averageSpeed*1000)|0)/1000;
var description = " (" + roundedDistance + " km)";
description += "<br>Average speed: " + roundedSpeed + " km/h";
description += "<br>" + nbPositions + " positions saved";
return description; }

Number.prototype.toRad = function() { return this * Math.PI / 180; }

function haversineDistance(lat1, long1, lat2, long2) {
var R = 6371; // km
var dLat = (lat2-lat1).toRad();
var dLon = (long2-long1).toRad();
var a = Math.sin(dLat/2) * Math.sin(dLat/2) + Math.cos(lat1.toRad()) *
            Math.cos(lat2.toRad()) * Math.sin(dLon/2) * Math.sin(dLon/2);
var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
var d = R * c;
return d; }
```

Quellcode-Abbildung 26: Funktionen zur Distanz- und Geschwindigkeits-Berechnung und Darstellung

Die Funktion positionInformation() rechnet die Positionswerte so um, dass neben dem Datum und der Laufzeit, nun auch Angaben über die zurückgelegten Kilometer, die Laufgeschwindigkeit und die insgesamt gespeicherten Positionen erhalten werden können. Für die notwendige Distanzberechnung wird eine Hilfsfunktion haversineDistance() verwendet (siehe Quellcode-Abb. 26), die über die jeweils übergebenen Längen- und Breitengrade einer Laufzeit die Distanz ermittelt und an positionInformation() ausliefert.

Auf eine nähere Erläuterung der Berechnungsformeln wird an dieser Stelle nicht weiter eingegangen, hierfür sei auf folgenden Link verwiesen, über den die Berechnungsformeln und der Code noch mal ausführlicher beschrieben werden:
`http://www.movable-type.co.uk/scripts/latlong.html`

Schritt 10: Maps-API verwenden

Um von der Laufzeiten-Ansicht zu der Maps-Ansicht zu gelangen (siehe Abb. 18), ist es notwendig das Google-Maps-API zu verwenden. Hierfür ist, wie schon erwähnt, die Einrichtung eines kostenlosen Google-Accounts notwendig, um den API-Key zu erhalten (`http://code.google.com/apis/maps/signup.html`).
Hat man diesen API-Key erhalten, wird er in „index.html“ implementiert:

```
<script type="text/JavaScript" src="gears_init.js"></script>
<script src= "http://maps.google.com/maps?file=api&amp;v=2&amp;key=YOURKEY"
type="text/JavaScript"></script>
<script type="text/JavaScript" src="model.js"></script>
```

Zusätzlich muss ein Aufruf zur unload-Funktion hinzugefügt werden, um die Javascript-Funktionen des Maps-APIs bereitzustellen:

```
<body onload="main()" onunload="GUnload()" >
```

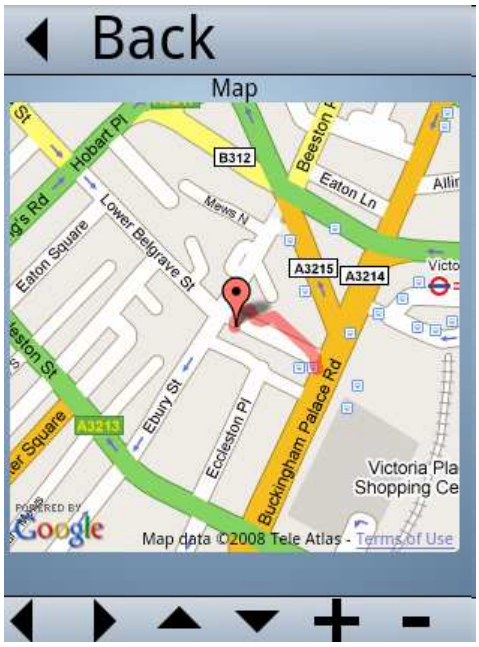


Abbildung 18: Maps-Ansicht

Um eine Google-Map überhaupt darstellen zu können, muss zudem noch ein neuer HTML-Bereich in „index.html“ definiert werden (siehe Quellcode-Abb. 27).
Dieser neue Bereich nennt sich „location“ und beinhaltet im unteren Teil lauter onclick-Funktionen zum links, rechts, unten und oben verschieben der Kartenansicht und zoomin- und zoomout-Funktionen zum vergrößern und verkleinern der Kartenansicht.

```
<div id="location" align="center"><div class="content">
<div id="mapLabel">Map</div>
<div id="map" style="width: 300px; height: 300px"></div></div>
<div class="menu" align="left">
<div class="back-button" onclick="go('mainScreen')">Back</div></div>
<div id="map-buttons" align="left">
<div class="button-left" onclick="mapLeft()"></div>
<div class="button-right" onclick="mapRight()"></div>
<div class="button-up" onclick="mapUp()"></div>
<div class="button-down" onclick="mapDown()"></div>
<div class="button-zoomin" onclick="mapZoomIn()"></div>
<div class="button-zoomout" onclick="mapZoomOut()"></div></div></div>
```

Quellcode-Abbildung 27: Neuer Bereich in index.html

Damit beim Wechsel auf die einzelnen HTML-Bereiche nicht andauernd die Maps-Karte angezeigt wird, muss diese ebenfalls in der hideAllScreens()-Funktion in model.js gelistet werden (siehe Quellcode-Abb. 28).

```
function hideAllScreens() {
hideDiv('mainScreen');
hideDiv('watch');
hideDiv('journeys');
hideDiv('location'); }
```

Quellcode-Abbildung 28: hideAllScreens modifizieren

Um zudem eine angemessene Vergrößerungssicht auf der Karte einzuhalten, wird der Mapzoom-Wert auf 15 eingestellt (siehe Quellcode-Abb. 29).

```
function main() {
global.startTime = null;
global.geo = google.gears.factory.create('beta.geolocation');
global.mapZoom = 15;
updateTime();
global.siteUrl = "http://YOURURL/index.html";
initDB();
installShortcut(); }
```

Quellcode-Abbildung 29: Mapzoom-Wert einstellen

Zudem müssen in model.js die in index.html benutzten Funktionen, zum verschieben und zoomen der Karte, definiert werden (siehe Quellcode-Abb. 30).

```
function mapLeft() { global.map.panDirection(1, 0); }
function mapRight() { global.map.panDirection(-1, 0); }
function mapUp() { global.map.panDirection(0, 1); }
function mapDown() { global.map.panDirection(0, -1); }
function mapZoomIn() { global.map.zoomIn(); }
function mapZoomOut() { global.map.zoomOut(); }
```

Quellcode-Abbildung 30: Maps-Funktionen definieren

Um nun von der Laufzeitenübersicht über das Anklicken eines Eintrags auf die entsprechende Maps-Darstellung zu gelangen, muß die Darstellungsfunktion „on_journeys“ entsprechend modifiziert werden (siehe Quellcode-Abb. 31).

```
html += "<div class='" + rowType + "'>";
html += "<div class='rowDesc' onclick='showMap(" + rowID + ")'>";
html += description + "</div>";
html += "<div class='rowImg' onclick='deleteRecord(" + rowID + ")'>";
html += "<img src='../images/delete.png'></div";     html += "</div>";
```

Quellcode-Abb. 31: on_journeys() modifizieren

Über onclick wird innerhalb der Funktion on_journeys() eine Funktion showMap() ausgeführt, die sich um die Darstellung über Google-Maps kümmert.

```
function showMap(rowID) {
hideAllScreens();
showDiv('location');
var lastPoint = null;
global.map = new GMap2(document.getElementById("map"));
var locations = new Array();
var rs = global.db.execute('SELECT Latitude, Longitude ' +
                           'FROM Positions WHERE TimeID = (?) ' +
                           'ORDER BY Date', [rowID]);
       while (rs.isValidRow()) {
                                var lat = rs.field(0);
                                var lon = rs.field(1);
                                var point = new GLatLng(lat, lon);
                                locations[locations.length] = point;
                                lastPoint = point; rs.next(); }
      if (lastPoint != null) {
                global.map.setCenter(lastPoint, global.mapZoom);
                var polyline = new GPolyline(locations, '#ff0000', 8);
                global.map.addOverlay(polyline);
                global.map.addOverlay(new GMarker(point)); } }
```

Quellcode-Abbildung 32: Die showMap()-Funktion

showMaps() arbeitet ähnlich der go()-funktion und versteckt erst alle HTML-Seitenbereiche über hideAllScreens und zeigt dann den Bereich „location“ an.
Im Anschluß daran wird dann eine Google-Map erzeugt, die aus der Positions-Tabelle die jeweiligen Längen- und Breitengrade verwendet und die einzelnen für eine Laufzeit gespeicherten Positionen als Laufstrecke anzeigt.

Schritt 11: Offline-Modus ermöglichen

Die RunningMan-Anwendung ist nun fast fertig, es müssen nur noch die erforderlichen Gears-Massnahmen getroffen werden, um die Anwendung offline fähig zu setzen.

Um die Ressourcen der Anwendung local zu speichern muß wieder ein Manifest-File zum listen aller benötigten Ressourcen angelegt werden.
Dieses Manifest-File wird als „manifest.json“ angelegt und die Version der Inhalte als „version 1.0“ benannt (siehe Quellcode-Abb. 33).

```
{ "betaManifestVersion": 1,
  "version": "version 1.0",
  "entries": [
              { "url": "index.html" },
              { "url": "gears_init.js" },
              { "url": "model.js" },
              { "url": "../run.css" },
              { "url": "../images/icon.png" },
              { "url": "../images/button-background.png" },
              { "url": "../images/background.png" },
              { "url": "../images/back-arrow.png" },
              { "url": "../images/go-arrow.png" },
              { "url": "../images/delete.png" },
              { "url": "../images/left.png" },
              { "url": "../images/right.png" },
              { "url": "../images/down.png" },
              { "url": "../images/up.png" },
              { "url": "../images/zoomIn.png" },
              { "url": "../images/zoomOut.png" } ] }
```

Quellcode-Abbildung 33: Manifest-File – „manifest.json“

Für das eigentliche Speichern wird dann noch eine Funktion captureOffline() in model.js hinzugefügt, die später mit der Main-Methode aufgerufen werden soll.
Über die Gears-APIs legt sie einen ManagedRessourceStore an, der das erstellte Manifest-File verwendet, um über checkForUpdate alle benötigten Files lokal zu speichern und automatisch zu aktualisieren.

```
function captureOffline() {
global.localserver = google.gears.factory.create("beta.localserver");
global.store = global.localserver.createManagedStore("Stopwatch");
global.store.manifestUrl = "manifest.json";
global.store.checkForUpdate();}
```

Quellcode-Abbildung 34: Die Funktion „captureOffline()“

Beim offline-verwenden der RunningMan-Anwendung tritt momentan aber noch ein Problem bei der Verwendung des Google-Maps-Apis ein.
Das Maps-API-Script wurde über den HTML-Header über eine Verlinkung geladen (siehe Seite 29).
Da diese Verlinkung aber auf eine Quelle außerhalb unserer Gears-Anwendung zeigt, kann Gears das File nicht für die offline-Verwendung speichern, da Gears nämlich eine sogenannte „same-location security policy“ verfolgt.
Gears kann nur Files local speichert, die sich im Verzeichnis der entsprechenden Anwendung befinden.
Da dies beim Maps-API-Script nicht der Fall ist, befindet sich beim offline verwenden von RunningMan das Maps-API-File nicht im Browser-Cache und die Anwendung wird geblockt.

Damit dieser Fall nicht eintritt ist es nun notwendig, das Maps-API im Hintergrund zu laden, also asynchron. Auf diese Weise kann RunningMan trotzdem offline ausgeführt werden, auch wenn das Laden des Maps-APIs fehlschlägt.

Für das asynchrone Laden des Maps-API wird eine weitere Funktion „loadMapsAPI()“ in model.js hinzugefügt (siehe Quellcode-Abb. 35).

```
function loadMapsAPI() {
var script = document.createElement("script");
var key = 'YOURKEY';
script.type = "text/javascript";
script.src = "http://maps.google.com/maps?file=api&v=2&async=2&key=" + key;
document.body.appendChild(script); }
```

Quellcode-Abbildung 35: Die Funktion loadMapsAPI()

Die Funktion loadMapsAPI() erzeugt ein neues Element des original Maps-API-Script und fügt es zur verwendung im Body der Anwendung ein.
Aufgerufen wird die Funktion dann, wie auch captureOffline(), in der Main-Methode:

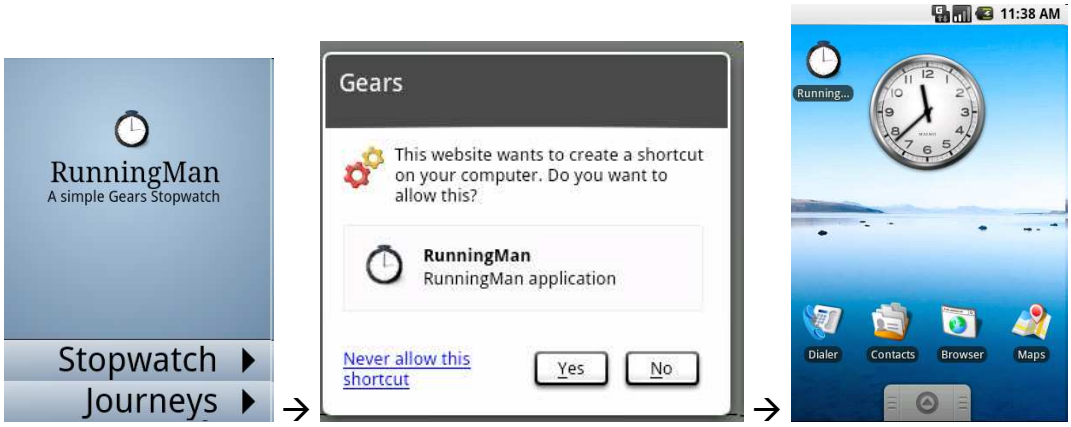
```
captureOffline();
loadMapsAPI();
```

Zum Schluß muss nun nur noch die Maps-API-Verlinkung im Header entfernt werden und die Kartenanzeigefunktion showMap() dahingehend modifiziert werden, dass sie eine Fehlermeldung ausgibt, wenn das Laden des Maps-API fehlschlägt:

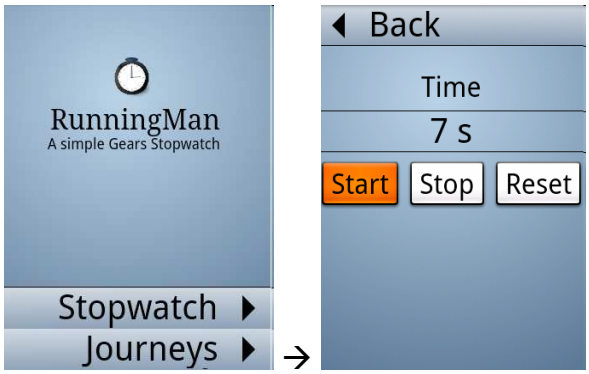
```
function showMap(rowID) {   hideAllScreens();
                            showDiv('location');
                            if (window["GMap2"] == null) {
                                alert('No Map object!'); return; }
```

Szenario

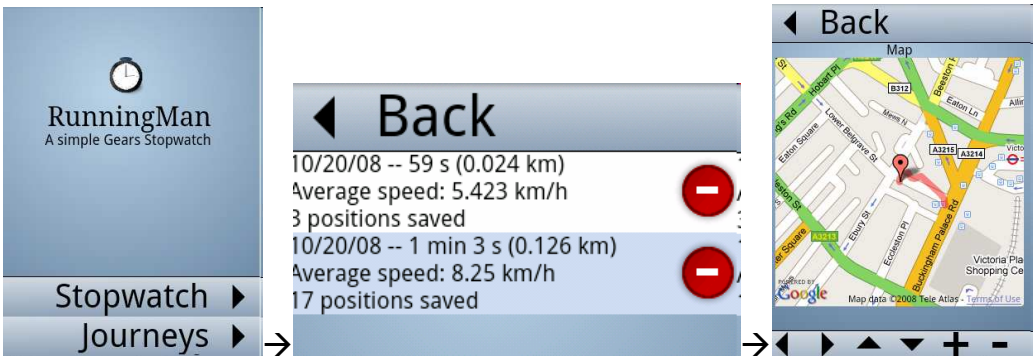
RunningMan sollte nun sowohl online als auch offline einsatzbereit seit.
Im Folgenden sei das Szenario für den optimalen Fall veranschaulicht:



Ein User hat Gears installiert und besucht die RunningMan-Site das erste mal online.
Er bekommt das Angebot eine Desktop-Verknüpfung für RunningMan anlegen zu können.
Dies macht er und hat ab da an einen entsprechenden Shortcut auf dem Android-Desktop.



Über den Shortcut ruft er die Anwendung auf und erkennt zwei Optionen:
„Stopwatch“ und „Journeys“. Klickt er auf „Stopwatch“ kann er diese nach belieben verwenden und ein bisschen dabei spazieren laufen…
Klickt er auf „Back“ und dann im Hauptmenü auf die zweite option „Journeys“, kann er sich seine verschiedenen gelaufenen Laufzeiten mit entsprechenden Informationen über Dauer, Distanz und Geschwindigkeit und die Anzahl der verschiedenen gespeicherten Zwischenpositionen ansehen und diese auch nach belieben wieder löschen.



Interessiert sich der User auch dafür, wie seine Laufroute auf einer Karte veranschaulicht aussieht, klickt er einfach auf den entsprechenden Laufzeiteintrag und sieht seine gelaufene Route in einer Google-Map.

Verwendet der User die RunningMan-Anwendung wenn er offline ist, so sollte auch dann die Anwendung lauffähig sein, bloß mit der Einschränkung, dass bei der Kartenanzeige eine Fehlermeldung auftaucht, da das Maps-API offline nicht geladen werden kann.
Die restlichen Funktionen wie Laufzeit und Geschwindigkeitsberechnung sollten jedoch uneingeschränkt möglich sein, da über das Geolocation-API von Gears, die Positionen nach wie vor über GPS im Android-Handy ermittelt werden können.

5. Sicherheit und Risiko bei Gears

Gears verwendet das sogenannte „same-origin“-Sicherheitskonzept, d.h. eine Website kann immer nur auf Ressourcen der gleichen Herkunfts-Quelle zugreifen.
Für eine Gears-Seite bedeutet dies, dass immer nur die eigene Datenbank geöffnet werden darf und auch nur die eigenen URLs im LocalServer gespeichert werden dürfen.
Zudem wird vor der ersten Ausführung von Gears, bzw. dem Runterladen der entsprechenden Ressourcen, ein Warndialog von Gears an den Benutzer ausgegeben, der erst bestätigt werden muss, bevor eine Gears-Seite die Gears-APIs verwenden kann.
Was die persönlichen Benutzer-Daten angeht, so sind diese nur über das benutzer-eigene Benutzer-Profil des jeweiligen Betriebssystems zugänglich bzw. über dessen Login.

Problematisch wird die Verwendung von Gears jedoch, wenn sich mehrere Personen den selben Login für ein Betriebssystem teilen, da sie sich dann auch das Zugriffsrecht auf die gespeicherten Gears-Daten teilen.
Solch eine Situation kann beispielsweise im privaten Haushalt oder in Internetcafes vorliegen.
Handelt es sich bei der Gears-Anwendung z.B. um eine Anwendung, die das Speichern, Ändern und/oder Abrufen von persönlichen Daten in oder aus der lokalen Gears Datenbank ermöglicht, so haben alle Benutzer desselben Profils Zugriffsrecht auf die Daten und können sie einsehen, ändern oder löschen.
Ein weiteres Risiko in Gears, auf das Google in seiner Gears-Doku hinweist, stellen SQL-Injection-Angriffe auf die Gears-Daten dar.
Maskiert ein Gears-Entwickler die Benutzereingaben in SQL-Statements nicht ausreichend, besteht die Gefahr, dass Daten über eingeschleuste SQL-Befehle ausspioniert oder verändert werden können.
(Wie genau das bei „same-origin policy“ möglich sein soll wird jedoch dort nicht erwähnt)
Zur Vorbeugung gegen SQL-Injection sollten Entwickler Benutzereingaben beim Programmieren von SQL-Statements nur über (?)-Austausch-Parameter übergeben.

Beispiel:
db.execute('insert into MyTable values (' + data + ')'); → zu unsicher
db.execute('insert into MyTable values (?)', data); → so ist es sicherer

6. Ausblick: HTML5 statt Gears

Seit kurzem werden gängige Standards wie HTML4 bzw. XHTML 1 durch einen neuen Webstandard namens HTML5 abgelöst.
Die HTML5-Spezifikation berücksichtigt sowohl Offline-Funktionalität als auch Standortbestimmung und muss im Gegensatz zu Gears nicht erst als Plugin installiert werden, da es einen Webstandard für Browser darstellt.
Die Weiterenwicklung von Funktionalitäten für Gears wurde mittlerweile seitens Google eingestellt und Google empfiehlt offiziell den Entwicklern von Webanwendungen, statt Gears den neuen Webstandard HTML5 zu nutzen. Auch wird es keine Gears-Versionen mehr für einige neuere Plattformen, wie z.B. den Safari-Browser unter Mac OS X 10.6 (Snow Leopard) geben.
Da Gears aber open-source ist, steht es immer noch anderen Programmierern frei, das Projekt selbst weiterzuführen und ihre Anwendungen Gears-fähig zu schreiben.
Gears ist zudem weiterhin erhältlich und wird wohl auch noch länger verwendet werden, da HTML5 zwar laufend weiterentwickelt wird, jedoch noch nicht in vollem Umfang für alle Browser kompatibel ist. Lediglich die neueste Version des Browsers „Google Chrome“ unterstützt HTML5 soweit, dass Gears nicht mehr notwendig ist.
Für ausgewählte ältere Browser ist Gears die Alternative zu HTML5 und sogar der aktuelle Internet Explorer- als auch Firefox-Browser werden zur Zeit noch mit Gears-Versionen unterstützt (Beispiel: Firefox 3.6).

Fazit:
HTML5 ist zwar stark für neuere Browser im Trend, jedoch sprechen immer noch die umständliche Portierung von Gears-Anwendungen zu HTML5 und die funktionellen Einschränkungen von HTML5 in gängigen Browsern für die Verwendung von Gears.
Zum Abschluss hier noch eine kleine Auflistung der kompatiblen HTML5-Funktionalitäten für verschiedene gängige Browser:

HTML5 Spezifikation	Gears Plugin	IE 6&7	IE 8	FF3	FF 3.5	Safari 4	Safari Mobile	Chrome	Opera
Offline	■			■	■	■	■	■	
Worker	■				■	■		■	
Local/ Session Storage			■		■	■	■		
Local Database	■					■	■	■	
Geolocation	■				■		■	■	
Video/Audio					■	■	■	■	■
Native JSON			■		■				
Cross Site XHR			■		■	■	■	■	
Cross-document messaging			■	■	■	■	■	■	
Canvas				■	■	■	■	■	■
DOM Selectors			■		■	■	■	■	
CSS Transform (Draft)					■	■	■	■	
CSS Media Queries					■	■	■	■	■
Drag&Drop vom Desktop	■								
Drag&Drop (HTML5)					■	■		■	
Blob Builder/ File Upload with XHR	■				■				

7. Anhang

A1 - Quellcodes zur „go_offline-Website“

„tutorial_manifest.json“

```
{
  "betaManifestVersion": 1,
  "version": "my_version_string",
  "entries": [
      { "url": "go_offline.html"},
      { "url": "go_offline.js"},
      { "url": "../gears_init.js"}
    ]
}
```

„go_offline.html“

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<script type="text/javascript" src="../gears_init.js"></script>
<script type="text/javascript" src="go_offline.js"></script>
<title>Enable Offline Usage</title>
<style type="text/css">
<!--
.style1 {
      color: #003399;
      font-style: italic;
      font-weight: bold;
      font-size: large;}
.style3 {
      color: #009933;
      font-weight: bold;
      font-style: italic;}
-->
</style>
</head>

<body onload="init()">
<h2>Getting offline-enabled documents with Gears </h2>
<p> </p>
<div>
<p class="style1">Status Message:
<span id="textOut" class="style3"></span></p>
</div>
<p><strong>Q:</strong> I want to see these documents when I'm not online!
What must I do?<br />
<strong>A:</strong> <a href="http://gears.google.com/">Install Gears</a> on
your computer and then click &amp;quot;Capture&amp;quot; to store the documents to
your computer. You can then access the URLs without a network connection.
</p><p>
<button onclick="createStore()" > Capture </button></p>
<p><strong>Q:</strong> I want to remove my offline access to these
documents. What must I do?<br />
```

```
<strong>A: </strong>Click &quot;Erase&quot; below to removed the
&quot;captured&quot; documents from your computer.  The documents will no
longer be available without a network connection. </p><p>
<button onclick="removeStore()" > Erase </button></p>
</body>
</html>
```

„go_offline.js“

```
// Change this to set the name of the managed resource store to create.
// You use the name with the createManagedStore, and removeManagedStore,
// and openManagedStore APIs. It isn't visible to the user.
var STORE_NAME = "my_offline_docset";

// Change this to set the URL of tha manifest file, which describe which
// URLs to capture. It can be relative to the current page, or an
// absolute URL.
var MANIFEST_FILENAME = "tutorial_manifest.json";
var localServer;
var store;

// Called onload to initialize local server and store variables
function init() {
  if (!window.google || !google.gears) {
    textOut("NOTE:  You must install Gears first.");
  } else {
    localServer = google.gears.factory.create("beta.localserver");
    store = localServer.createManagedStore(STORE_NAME);
    textOut("Yeay, Gears is already installed.");
  }
}

// Create the managed resource store
function createStore() {
  if (!window.google || !google.gears) {
    alert("You must install Gears first.");
    return;
  }

  store.manifestUrl = MANIFEST_FILENAME;
  store.checkForUpdate();

  var timerId = window.setInterval(function() {
    // When the currentVersion property has a value, all of the resources
    // listed in the manifest file for that version are captured. There is
    // an open bug to surface this state change as an event.
    if (store.currentVersion) {
      window.clearInterval(timerId);
      textOut("The documents are now available offline.\n" +
              "With your browser offline, load the document at " +
              "its normal online URL to see the locally stored " +
                          "version. The version stored is: " +
              store.currentVersion);
    } else if (store.updateStatus == 3) {
      textOut("Error: " + store.lastErrorMessage);
    }
  }, 500);
}
```

```
// Remove the managed resource store.
function removeStore() {
  if (!window.google || !google.gears) {
    alert("You must install Gears first.");
    return;
  }

  localServer.removeManagedStore(STORE_NAME);
  textOut("Done. The local store has been removed." +
          "You will now see online versions of the documents.");
}

// Utility function to output some status text.
function textOut(s) {
 var elm = document.getElementById("textOut");
  while (elm.firstChild) {
    elm.removeChild(elm.firstChild);
  }
  elm.appendChild(document.createTextNode(s));
}
```

„gears_init.js“

```
(function() {
  // We are already defined. Hooray!
  if (window.google && google.gears) {
    return;
  }

  var factory = null;

  // Firefox
  if (typeof GearsFactory != 'undefined') {
    factory = new GearsFactory();
  } else {
    // IE
    try {
      factory = new ActiveXObject('Gears.Factory');
 // privateSetGlobalObject is only required and supported on IE Mobile on
 // WinCE.
      if (factory.getBuildInfo().indexOf('ie_mobile') != -1) {
        factory.privateSetGlobalObject(this);
      }
    } catch (e) {
      // Safari
      if ((typeof navigator.mimeTypes != 'undefined')
           && navigator.mimeTypes["application/x-googlegears"]) {
        factory = document.createElement("object");
        factory.style.display = "none";
        factory.width = 0;
        factory.height = 0;
        factory.type = "application/x-googlegears";
        document.documentElement.appendChild(factory);
      }
    }
  }
```

```
// *Do not* define any objects if Gears is not installed. This mimics the
// behavior of Gears defining the objects in the future.
  if (!factory) {
    return;
  }

// Now set up the objects, being careful not to overwrite anything.
// Note: In Internet Explorer for Windows Mobile, you can't add properties
// to the window object. However, global objects are automatically added as
// properties of the window object in all browsers.
  if (!window.google) {
    google = {};
  }

  if (!google.gears) {
    google.gears = {factory: factory};
  }
})();
```

A2 - Quellcodes zu „RunningMan“

„index.html“

```
<html>
<head id="head">
  <title>RunningMan</title>
  <link rel="stylesheet" type="text/css" href="../run.css" />
  <script type="text/javascript" src="../../gears_init.js"></script>
  <script type="text/javascript" src="model.js"></script>
  <meta name="viewport" content="width=320; user-scalable=false">
</head>
<body onload="main()" onunload="GUnload()">
  <div id="mainScreen" align="center">
    <div id="title">
      <img src="../images/icon.png" width="48" height="48">
      <br>RunningMan<div id="subtitle">A simple Gears Stopwatch</div>
    </div>
    <div class="main-menu" align="right">
      <div class="go-button" onclick="go('watch')">Stopwatch</div>
      <div class="go-button" onclick="go('journeys')">Journeys</div>
    </div>
  </div>
  <div id="watch" align="center">
    <div class="content">
      <div id="timeTitle">Time</div>
      <div id="timeDisplay"></div>
      <input type="button" onclick="startRun()" value="Start"></input>
      <input type="button" onclick="stopRun()" value="Stop"></input>
      <input type="button" onclick="resetRun()" value="Reset"></input>
    </div>
    <div class="menu" align="left">
      <div class="back-button" onclick="go('mainScreen')">Back</div>
    </div>
  </div>
  <div id="journeys">
    <div class="content">
      <div id="journeysContent"><h2>No Journeys Yet!</h2></div>
    </div>
    <div class="menu" align="left">
      <div class="back-button" onclick="go('mainScreen')">Back</div>
    </div></div>
```

```
  <div id="location" align="center">
    <div class="content">
      <div id="mapLabel">Map</div>
      <div id="map" style="width: 300px; height: 300px"></div>
    </div>
    <div class="menu" align="left">
      <div class="back-button" onclick="go('mainScreen')">Back</div>
    </div>
    <div id="map-buttons" align="left">
      <div class="button-left" onclick="mapLeft()"></div>
      <div class="button-right" onclick="mapRight()"></div>
      <div class="button-up" onclick="mapUp()"></div>
      <div class="button-down" onclick="mapDown()"></div>
      <div class="button-zoomin" onclick="mapZoomIn()"></div>
      <div class="button-zoomout" onclick="mapZoomOut()"></div>
    </div>
  </div>
</body>
</html>
```

„model.js“

```
var global = new Object();

/*
 * Main function
 */
function main() {
  global.startTime = null;
  global.geo = google.gears.factory.create('beta.geolocation');
  global.mapZoom = 15;
  updateTime();
  global.siteUrl =
"http://code.google.com/apis/gears/samples/running_man/step8/index.html";
  initDB();
  installShortcut();
  captureOffline();
  loadMapsAPI();
}

/*
 * Loads the Google Maps API asynchronously
 */
function loadMapsAPI() {
  var script = document.createElement("script");
  var key =
'ABQIAAAAsZ2C8blcRF_NO1nRClSs4xT7_rQBWrRQBxIxHmD38f4UhmkYzRT_oAn4JlE-
tnpyS2NUN4Zh0L5PgQ';
script.type = "text/javascript";
script.src = "http://maps.google.com/maps?file=api&v=2&async=2&key=" +key;
document.body.appendChild(script);
}
/*
 * Capture resources for offline mode
 */
function captureOffline() {
  global.localserver = google.gears.factory.create("beta.localserver");
  global.store = global.localserver.createManagedStore("Stopwatch");
  global.store.manifestUrl = "manifest.json";
  global.store.checkForUpdate();
}
```

```
/*
 * Initialize the Database
 */
function initDB() {
  global.db = google.gears.factory.create('beta.database');
  global.db.open('stopwatch');
  global.db.execute('CREATE TABLE IF NOT EXISTS Preferences ' +
             '(Name text, Value int)');
  global.db.execute('CREATE TABLE IF NOT EXISTS Times ' +
             '(StartDate int, StopDate int, Description text)');
  global.db.execute('CREATE TABLE IF NOT EXISTS Positions ' +
             '(TimeID int, Date int, Latitude float, Longitude float, ' +
             'Accuracy float, Altitude float, AltitudeAccuracy float)');
}

function getPreference(name) {
var result = false;
var rs = global.db.execute('SELECT Value FROM Preferences WHERE Name =
(?)', [name]);

  if (rs.isValidRow()) {
    result = rs.field(0);
  }
  rs.close();
  return result;
}

function setPreference(name, value) {
  global.db.execute('INSERT INTO Preferences VALUES (?, ?)', [name,
value]);
}

/*
 * Install shortcut
 */
function installShortcut() {
  if (getPreference('Shortcut') == false) {
    var desktop = google.gears.factory.create('beta.desktop');
    desktop.createShortcut("RunningMan", global.siteUrl,
      { "48x48" : "../images/icon.png" }, "RunningMan Step8");
    setPreference('Shortcut', true);
  }
}

/*
 * Timers functions
 */

function startRun() {
  global.updateInterval = setInterval("updateTime()", 1000);
  global.startTime = new Date();
  var time = global.startTime.getTime();
  global.db.execute('INSERT INTO Times (StartDate) VALUES (?)', [time]);
  global.currentTimeID = global.db.lastInsertRowId;
  global.currentGeoWatcher = global.geo.watchPosition(function (position) {
    global.db.execute('INSERT INTO Positions (TimeID, Date, Latitude, ' +
               'Longitude, Accuracy, Altitude, AltitudeAccuracy) ' +
               'VALUES (?, ?, ?, ?, ?, ?, ?)',
               [global.currentTimeID, position.timestamp.getTime(),
               position.latitude, position.longitude, position.accuracy,
               position.altitude, position.altitudeAccuracy]);
    }, null, { "enableHighAccuracy" : true});}
```

```
function stopRun() {
  clearInterval(global.updateInterval);
  var stopDate = new Date();
  var time = stopDate.getTime();
  global.db.execute('UPDATE Times SET StopDate = (?) ' +
                'WHERE ROWID = (?)', [time, global.currentTimeID]);
  if (global.currentGeoWatcher != null) {
    global.geo.clearWatch(global.currentGeoWatcher);
    global.currentGeoWatcher = null;
  }
}

function resetRun() {
  stopRun();
  global.startTime = null;
  updateTime();
}

/*
 * Show journeys
 */
function on_journeys() {
  var rows = 0;
  var html = "";

  var rs = global.db.execute('SELECT ROWID, Description FROM Times');

  while (rs.isValidRow()) {
    var rowID = rs.field(0);
    var description = rs.field(1);

   if (description == null) {
      description = createDescription(rowID);
   }

    var rowType;
    if (rows % 2 == 0) {
      rowType = "rowA";
    } else {
      rowType = "rowB";
    }
    rows++;

    html += "<div class='" + rowType + "'>";
    html += "<div class='rowDesc' onclick='showMap(" + rowID + ")'>" +
description + "</div>";
    html += "<div class='rowImg' onclick='deleteRecord(" + rowID + ")'>";
    html += "<img src='../images/delete.png'></div";
    html += "</div>";

    rs.next();
  }

  if (html != "") {
    var elem = document.getElementById('journeysContent');
    elem.innerHTML = html;
  }
}
```

```
/*
 * Map navigation functions
 */

function mapLeft() {
  global.map.panDirection(1, 0);
}

function mapRight() {
  global.map.panDirection(-1, 0);
}

function mapUp() {
  global.map.panDirection(0, 1);
}

function mapDown() {
  global.map.panDirection(0, -1);
}

function mapZoomIn() {
  global.map.zoomIn();
}

function mapZoomOut() {
  global.map.zoomOut();
}

/*
 * Extract the positions information to plot
 * them on a map
 */
function showMap(rowID) {
  hideAllScreens();
  showDiv('location');
  if (window["GMap2"] == null) {
    alert('No Map object!');
    return;
  }

  var rs = global.db.execute('SELECT Latitude, Longitude ' +
                             'FROM Positions WHERE TimeID = (?) ' +
                             'ORDER BY Date', [rowID]);

  var lastPoint = null;
  global.map = new GMap2(document.getElementById("map"));
  var locations = new Array();
  while (rs.isValidRow()) {
    var lat = rs.field(0);
    var lon = rs.field(1);
    var point = new GLatLng(lat, lon);
    locations[locations.length] = point;
    lastPoint = point;
    rs.next();
  }

  if (lastPoint != null) {
    global.map.setCenter(lastPoint, global.mapZoom);
    var polyline = new GPolyline(locations, '#ff0000', 8);
    global.map.addOverlay(polyline);
    global.map.addOverlay(new GMarker(point));
  }
}
```

```
/*
 * Utility functions to compute the distance between
 * two coordinates
 */
Number.prototype.toRad = function() {  // convert degrees to radians
  return this * Math.PI / 180;
}

function haversineDistance(lat1, long1, lat2, long2) {
  // from http://www.movable-type.co.uk/scripts/latlong.html
  var R = 6371; // km
  var dLat = (lat2-lat1).toRad();
  var dLon = (long2-long1).toRad();
  var a = Math.sin(dLat/2) * Math.sin(dLat/2) +
          Math.cos(lat1.toRad()) * Math.cos(lat2.toRad()) *
                  Math.sin(dLon/2) * Math.sin(dLon/2);
  var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
  var d = R * c;
  return d;
}

/*
 * Returns informations based on the associated positions
 */
function positionInformation(rowID) {
  var distance = 0;
  var prevLat = null;
  var prevLon = null;
  var firstTime = 0;
  var lastTime = 0;
  var nbPositions = 0;
  var rs = global.db.execute('SELECT Latitude, Longitude, Date ' +
                             'FROM Positions WHERE TimeID = (?) ' +
                             'ORDER BY Date', [rowID]);

  while (rs.isValidRow()) {
    nbPositions++;
    var lat = rs.field(0);
    var lon = rs.field(1);
    var date = rs.field(2);

    if (firstTime != 0) {
      distance += haversineDistance(prevLat, prevLon, lat, lon);
    } else {
      firstTime = date;
    }
    prevLat = lat;
    prevLon = lon;
    lastTime = date;
    rs.next();
  }

  var secTime = (lastTime - firstTime) / 1000;
  var averageSpeed = ((distance * 3600) / secTime);
  var roundedDistance = (((distance*1000)|0)/1000);
  var roundedSpeed = ((averageSpeed*1000)|0)/1000;

  var description = " (" + roundedDistance + " km)";
  description += "<br>Average speed: " + roundedSpeed + " km/h";
  description += "<br>" + nbPositions + " positions saved";

  return description;
}
```

```
/*
 * Create the text displayed in the journeys pane for one record
 */
function createDescription(rowID) {
  var description = "";
  var rs = global.db.execute('SELECT StartDate, StopDate FROM Times ' +
                             'WHERE ROWID = (?)', [rowID]);

  if (rs.isValidRow()) {
    var sDate = rs.field(0);
    var eDate = rs.field(1);

    var time = (((eDate - sDate)/1000)|0); // elapsed time in seconds
    var startDate = new Date();
    startDate.setTime(sDate);

    description = startDate.toLocaleDateString() + " -- ";
    description += formatTime(time);
    description += positionInformation(rowID);

    global.db.execute('UPDATE Times SET Description = (?) ' +
                      'WHERE ROWID = (?)', [description, rowID]);
  }

  return description;
}

/*
 * Delete a recorded time
 */
function deleteRecord(rowID) {
  var answer = confirm("Delete this run?");
  if (answer) {
    global.db.execute('DELETE FROM Times WHERE ROWID = (?)', [rowID]);
    global.db.execute('DELETE FROM Positions where TimeID = (?)', [rowID]);
    go('journeys');
  }
}

/*
 * display stopwatch time value
 */
function updateTime() {
  var time = 0;
  if (global.startTime != null) {
    time = (new Date()).getTime() - global.startTime;
    time = (time/1000)|0;
  }
  var timeDiv = document.getElementById("timeDisplay");
  timeDiv.innerHTML = formatTime(time);
}
```

```
/*
 * Format a time value given in seconds
 */
function formatTime(aTime) {
  var seconds = aTime % 60;
  var minutes = ((aTime / 60) |0) % 60;
  var hours = (aTime / 3600) |0;
  var time = "0";
  if (seconds > 0) {
    time = seconds + " s";
  }
  if (minutes > 0) {
    time = minutes + " min " + time;
  }
  if (hours > 0) {
    time = hours + " h " + time;
  }
  return time;
}

/*
 * Navigation functions
 */

function go(name) {
  hideAllScreens();
  var functionName = "on_" + name;
  showDiv(name);
  if (window[functionName] != null) {
    window[functionName]();
  }
}

function showDiv(name) {
  var elem = document.getElementById(name);
  if (elem) {
    elem.style.display="block";
  }
}

function hideDiv(name) {
  var elem = document.getElementById(name);
  if (elem) {
    elem.style.display="none";
  }
}

function hideAllScreens() {
  hideDiv('mainScreen');
  hideDiv('watch');
  hideDiv('journeys');
  hideDiv('location');
}
```

„manifest.json“

```
{
  // version of the manifest file format
  "betaManifestVersion": 1,

  // version of the set of resources described in this manifest file
  "version": "version 1.0",

  // URLs to be cached (URLs are given relative to the
  // manifest URL)
  "entries": [
    { "url": "index.html" },
    { "url": "gears_init.js" },
    { "url": "model.js" },
    { "url": "../run.css" },
    { "url": "../images/icon.png" },
    { "url": "../images/button-background.png" },
    { "url": "../images/background.png" },
    { "url": "../images/back-arrow.png" },
    { "url": "../images/go-arrow.png" },
    { "url": "../images/delete.png" },
    { "url": "../images/left.png" },
    { "url": "../images/right.png" },
    { "url": "../images/down.png" },
    { "url": "../images/up.png" },
    { "url": "../images/zoomIn.png" },
    { "url": "../images/zoomOut.png" }
  ]
}
```

„run.css“

```
html {
overflow-y: hidden;
}

body {
 margin: 0;
 padding: 0;
 background: url(images/background.png);
}

.menu {
 position: fixed;
 top: 0;
 width: 100%;
 margin: 0;
 padding: 0;
 background: url(images/button-background.png);
 border-bottom: 1px solid black;
 border-top: 1px solid black;
}
```

```
.main-menu {
 position: fixed;
 bottom: 0;
 width: 100%;
 margin: 0;
 padding: 0;
 background: url(images/button-background.png);
 border-bottom: 1px solid black;
 border-top: 1px solid black;
}

.content {
  margin-top: 48px;
}

.rowA {
  width: 100%;
  background: white;
  display: table-row;
}

.rowB {
  width: 100%;
  background: #ccddf1;
  display: table-row;
}

.rowDesc {
  width: 100%;
  vertical-align: middle;
  display: table-cell;
}

.rowImg {
  width: 48px;
  vertical-align: middle;
  display: table-cell;
}

#timeTitle {
  padding-top: 1em;
  font-size: 24pt;
}

#timeDisplay {
  font-size: 30pt;
  border-top: 1px solid black;
  border-bottom: 1px solid black;
  margin-top: 8px;
  margin-bottom: 8px;
}

input {
  font-size:24pt;
  padding: 6pt;
}

.button-left {
  width: 32px;
  height: 32px;
  background: url(images/left.png);
  background-repeat: no-repeat;
  display:table-cell;}
```

```
.button-right {
  width: 32px;
  height: 32px;
  background: url(images/right.png);
  background-repeat: no-repeat;
  display:table-cell;
}

.button-up {
  width: 32px;
  height: 32px;
  background: url(images/up.png);
  background-repeat: no-repeat;
  display:table-cell;
}

.button-down {
  width: 32px;
  height: 32px;
  background: url(images/down.png);
  background-repeat: no-repeat;
  display:table-cell;
}

.button-zoomin {
  width: 32px;
  height: 32px;
  background: url(images/zoomIn.png);
  background-repeat: no-repeat;
  display:table-cell;
}

.button-zoomout {
  width: 32px;
  height: 32px;
  background: url(images/zoomOut.png);
  background-repeat: no-repeat;
  display:table-cell;
}

#map-buttons {
 position: fixed;
 bottom: 0;
 width: 100%;
 height: 32px;
 margin: 0;
 padding: 0;
 background: url(images/button-background.png);
 border-bottom: 1px solid black;
 border-top: 1px solid black;
 display:table;
}


.back-button {
  font-size: 30pt;
  padding-left: 1.5em;
  background: url(images/back-arrow.png);
  background-repeat: no-repeat;
}
```

```
.go-button {
  height: 48px;
  font-size: 30pt;
  padding-right: 1.5em;
  background: url(images/go-arrow.png);
  background-repeat: no-repeat;
  background-position: right;
}

.go-button:hover {
  background-color: white;
}

.back-button:hover {
  background-color: white;
}

#title {
  font-family: Times, Serif;
  font-size: 24pt;
  padding-top: 2em;
  padding-bottom: 3em;
}

#subtitle {
  font-family: Helvetica, Sans-Serif;
  font-size: 13pt;
}


#mainScreen {
  display: block;
}

#watch {
  display: none;
}

#journeys {
  display: none;
}

#location {
  display: none;
}

#preferences {
  display: none;
}

#locations {
}
```

8. Literatur

Gears-Homepage: 17.03.2010
<http://gears.google.com>

Gears-Website-Tutorial: 17.03.2010
<http://code.google.com/intl/de-DE/apis/gears/tutorial.html>

Gears-RunningMan-Tutorial: 17.03.2010
<http://code.google.com/intl/
de-DE/apis/gears/articles/running_man.html>

Gears-Artikel: 17.03.2010
<http://www.linux-magazin.de/Online-Artikel/
Anwendungen-fuer-Online-und-Offline-mit-Google-Gears-Teil-
1/%28offset%29/4>

<http://www.puremedia-online.de/fileadmin/Publikationen/
2008-04-17_web20kongress_offline.pdf>

Gears-Blob-Beispiel : 17.03.2010
<http://code.google.com/p/gears/issues/detail?id=994>

Gears vs HTML5 –Artikel: 17.03.2010
<http://www.golem.de/0912/71588.html>

<http://www.pcwelt.de/start/dsl_voip/online/news/
2106837/google-setzt-auf-html5-statt-google-gears/>

<http://almaer.com/blog/gears-as-a-bleeding-edge-html-5-
implementation>

<http://www.pocketbrain.de/newsticker/news/2713-google-html5-
statt-gears.html>

<http://spreadsheets.google.com/ccc?key=rp3FEESll4-
WG1CwPL6hpPw&hl=en>

<http://www.golem.de/1002/73274.html>

<http://gearsblog.blogspot.com/2010/02/hello-html5.html>