SEVENTH FRAMEWORK PROGRAMME

ICT PPP

Future Internet

The Environmental Observation Web and its Service Applications within the Future Internet

FP7-284898

Collaborative project

Mobile Application Development Documentation

Clemens Bernhard Geyer

Current release date: 2012-05-11

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 284898

Document Control Page

Title	Mobile Application Development Documentation
Creator	Clemens Bernhard Geyer (AIT)
Description	This documentation gives an overview of the technologies and architectures used and implemented for the mobile prototypes of WP1 and WP2.
Publisher	ENVIROFI Consortium
Contributors	
Creation date	2012-04-12
Type	Text
Language	en-GB
Rights	copyright “ENVIROFI Consortium”
Audience	☒ internal ☐ public ☒ restricted
Review status	☒ Draft ☐ WP leader accepted ☐ Technical Manager accepted ☐ Coordinator accepted
Action requested	☐ to be revised by Partners ☐ for approval by the WP leader ☐ for approval by the Technical Committee ☐ for approval by the Project Coordinator
Requested deadline	

Revision History

Version	Date	Modified by	Comments
0.1	2012-04-12	Clemens Bernhard Geyer (AIT)	Initial Version
0.2	2012-05-11	Clemens Bernhard Geyer (AIT)	Completed Sencha Touch 2 Documentation
...	...	...	...
1.0	<<dd/mm/yyyy>>	<<name>> (<<organization>>)	xxx

Table of Contents

Index of Figures

Index of Tables

Glossary

The glossary of terms used in this deliverable can be found in the public document "ENVIROFI_Glossary.pdf" available at: `http://www.envirofi.eu/`

Abbreviations and Acronyms

Abbreviation / Acronym	Description
GEO	Group on Earth Observations
GEOSS	Global Earth Observation System of Systems
GMES	Global Monitoring for Environment and Security
FI-PPP	Future Internet Public-Private Partnership
INSPIRE	Infrastructure for Spatial Information in Europe
MDAF	Mobile Data Acquisition Framework
OGC	Open Geospatial Consortium
OMG	Object Management Group
REST	Representational State Transfer
RM-ODP	Reference Model for Object Distributed Processing
SDI	Spatial Data Infrastructure
SDK	Software Development Kit
SOA	Service Oriented Architecture
SoS	System of Systems
SoSE	System of Systems Engineering
SWE	Sensor Web Enablement
UML	Unified Modelling Language
VGI	Volunteered Geographic Information
VP	Viewpoint
W3C	World Wide Web Consortium
XML	Extensible Mark-up Language

Table 1: Abbreviations and Acronyms

Executive Summary

<<Executive summary>>

1 Introduction

tbd <<Deliverable introduction (1 page)>>

2 Overview of Used Technologies

The implementation of the prototypes for WP1 and WP2 makes use of several technologies and libraries. To provide an application which can easily be deployed on multiple platforms (e.g., Android and iOS), the main part is implemented in HTML5 and JavaScript. Nevertheless, some parts of the application have to be implemented natively or make use of native libraries. Moreover, one part of the storage of the application is located on a server and has to provide well-defined interfaces.

The implementation of the prototype involves the following technologies:

- Android SDK and Android Eclipse Plugins
- PhoneGap 1.2.0 for Android
- Sencha Touch 2 and Sencha build
- OpenLayers 2.11
- Apache CouchDB 1.1.0 and 1.2.0

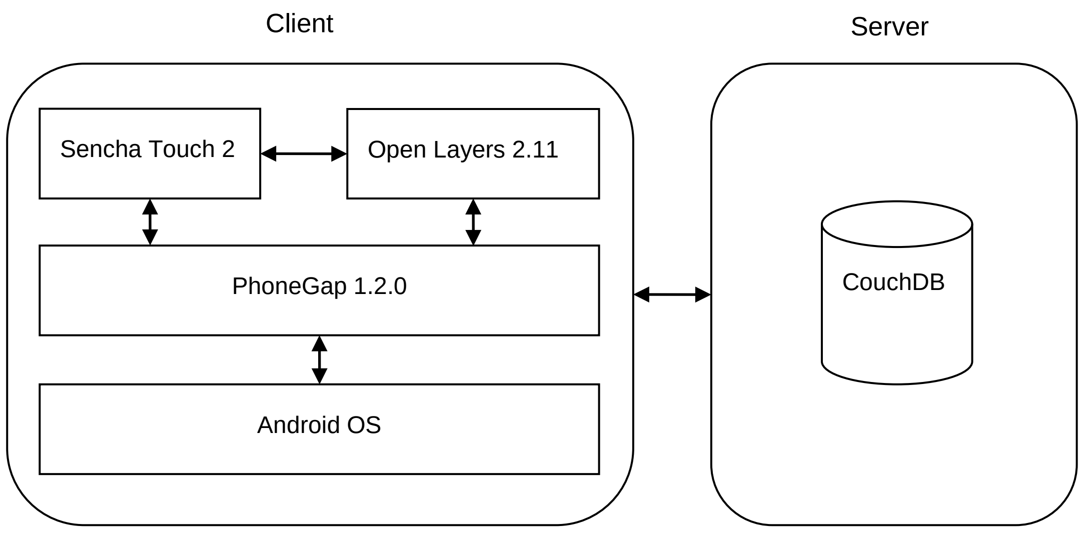


Figure 1: Overview of used technologies and their relationships on the client and the server.

2.1 The Android SDK

2.1.1 The Android SDK Tools

The core of the complete building process of Android applications is based on the Android software development kit (SDK) tools. They are available for Windows, MacOS X and Linux and may be retrieved from `http://developer.android.com/sdk/index.html`. The tools require a working installation of the Java runtime environment and the Java development kit. After the installation, you can start the *SDK Manager*, which is usually located in <INSTALL-PATH>/tools/android. Make sure you install the tools for *all* versions of Android you want to create applications for. Otherwise, it is not possible to create an APK file for a certain version or start a simulator. Nevertheless, it is usually sufficient to only download the oldest and the newest version, because older packages may easily be installed on newer versions of Android. This is not true the other way round. A good trade-off is to support devices with Android 2.1.

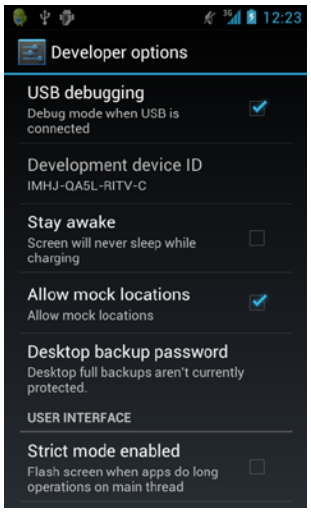
There are two types of the SDK tools: platform-dependent and platform-independent tools. The latter type includes the SDK manager, which is implemented in Java and is responsible for downloading and updating of required libraries and tools. Platform-dependent tools include the *Android Virtual Device*

(AVD) Manager and the *Android Debug Bridge (ADB)*. The latter may be used to list connected devices, install new applications etc. You can run the tools either in a command line shell or via a graphical user interface integrated in Eclipse. You can find platform-independent tools in the *tools* directory of your Android SDK installation; the platform-dependent tools are located in the *platform-tools* directory.

Known issue on Windows: If you want to update libraries or install new tools with the Android SDK Manager, it might happen that the specified package cannot be installed because you do not have write permissions in the installation directory of the SDK tools. Simply provide your current user full access to the installation directory or start the SDK manager as administrator to avoid this problem.

2.1.2 Driver Installation for USB Debugging Interface on Windows

If you want to develop on Windows, you have to install a driver for the USB debugging interface. These are available on `http://developer.android.com/sdk/win-usb.html` or may be installed via the SDK Manager. If you cannot find an appropriate driver on the described page, check the webpage of the manufacturer of your device. You do not have to install these drivers if you are working on either MacOS or Linux. In any case, make sure you enable the debugging option in your system settings (see Figure 2) of your phone. Note that this menu may look differently on other devices or versions of Android.



12:23

Developer options

USB debugging ✓
Debug mode when USB is connected

Development device ID
IMHJ-QA5L-RITV-C

Stay awake ☐
Screen will never sleep while charging

Allow mock locations ✓
Allow mock locations

Desktop backup password
Desktop full backups aren't currently protected.

USER INTERFACE

Strict mode enabled ☐
Flash screen when apps do long operations on main thread

Figure 2: Enabling USB debugging such that you can monitor your device with the Android SDK tools.

To check whether your operating system is able to communicate with your smartphone, plug in the device, open a terminal, change to the platform-tools directory of your Android SDK installation and execute the following command:

```
adb devices
```

If you get a serial number as output, your device has been recognized successfully. When working on virtual machines, it is only necessary to install drivers for your guest system. Note that there are some problems when using VirtualBox: some devices may currently not be connected to the guest system such that they are recognized by the android tools (e.g., the Samsung Nexus S does not work). Nevertheless, this is not true for all devices and there are no problems when using VMWare.

2.1.3 Android Eclipse Plugins

Download and install the latest classic version of Eclipse, which is available on `http://www.eclipse.org/downloads/`. Go to `http://developer.android.com/sdk/eclipse-adt.html` and download the latest version of the *Android Development Tools (ADT)*. Note that the specified link at the Android homepage to register the ADT repository in Eclipse is currently broken so that you have to install it locally. Start Eclipse and go to "Help → Install New Software…". Press the "Archive…" button and select the previously downloaded ADT archive file. Accept the license agreement and install all provided packages. Do not forget to restart Eclipse after you have installed the

ADT plugin.

Before you start working with the Eclipse tools, make sure that the path to the Android SDK is specified correctly: Go to “Window → Preferences → Android” and set the SDK location field in case it is not set correctly.

You can open the virtual devices manager by going to “Window → AVD Manager”. You can either create a new Android emulator or start an existing one. Note that you can only create emulators for which you have installed all required tools.

2.1.4 Android Project Structure

If you create an empty Android project, the following documents and folders will be created:

- src: contains the (Java) source code of the current project. There, you can also find the code of your main activity.
- gen: contains the building rules and calculated addresses of needed resources (e.g., memory locations of string constants).
- assets: contains all kind of needed resource files (e.g., sound files) and is used by PhoneGap as default directory for all HTML and JavaScript files.
- bin: contains the compiled (Java) code of the current project.
- libs (optional): contains Java libraries which will be linked to the final application (e.g., PhoneGap).
- res: contains all icons used in the current application in different resolutions. Moreover, the standard layout of the application and several String constants may be saved in an XML file.
- AndroidManifest.xml: contains all relevant high-level information of the application, e.g., which is the minimal required Android version or which permissions the application needs.
- The remaining files of the project folder are created and modified automatically by the Eclipse tools and contain configuration information of the current project.

2.1.5 Debugging on Android

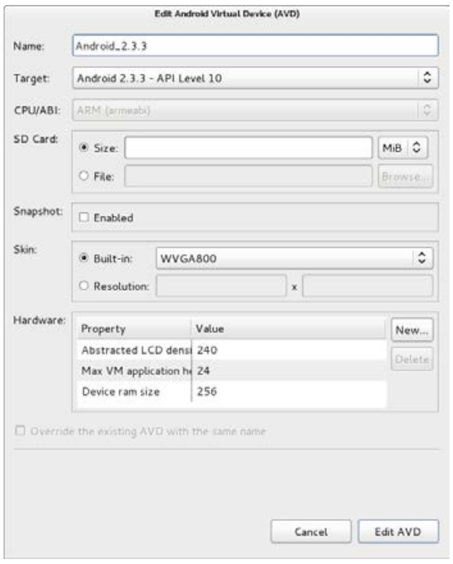
Theoretically, it is possible to get all relevant information of a running device by using ADB:

```
adb –d logcat
```

shows a complete logfile of the currently connected device. Although this view provides a lot of information, it is not very useful for finding out specific messages of a single program. When using Eclipse tools, you can activate *logcat* by going to “Window → Show View → Other… → Android → LogCat”. A small tab at the bottom of your current Eclipse window is now available. You can specify several filter parameters to only view messages which you are really interested in.

If you want to enable the complete debugging view, you have to go to “Window → Open Perspective → Other… → DDMS”. There, you have an overview of all active processes on your device as well as information about the heap and saved files. Furthermore, you can provide multiple settings for an emulator, e.g., the current GPS location.

When using an emulator for debugging, make sure that you add all hardware modules which you want to test or make use of in your application: Figure 3 shows the properties dialog of an emulator; you can simply add a new feature by pressing the “New…” button in the “Hardware” section. You can now select, e.g., a GPS module which will be added to the virtual device. Note that setting a faked location of an emulator does not have any effect if the GPS service is not enabled.



Name: Android_2.3.3

Target: Android 2.3.3 - API Level 10

CPU/ABI: ARM (armeabi)

SD Card: (●) Size: MiB ( ) File:

Snapshot: ☐ Enabled

Skin: (●) Built-in: WVGA800 ( ) Resolution: x

Hardware:

Property	Value
Abstracted LCD densi	240
Max VM application he	24
Device ram size	256

☐ Override the existing AVD with the same name

Figure 3: Configuration menu of the AVD Manager. You can add new (hardware) features to the selected emulator by clicking on the “New…” button in the section “Hardware” of the dialog window.

2.2 PhoneGap

2.2.1 General Features

PhoneGap started as an open-source project and is now maintained by Adobe. However, the developers assure that PhoneGap will remain free to use. It is a collection of libraries which allow programmers to access the hardware of a device via JavaScript. Thus, the main application can be implemented using HTML5 and JavaScript while the platform-dependent part is provided by PhoneGap. Such, it is possible to gain access to the camera, GPS sensor and even local files of the phone. Moreover, PhoneGap can be easily extended by so called *plugins* such that platform-dependent hardware may be available, too. The currently used version of PhoneGap is 1.2.0, but using a newer version should not cause any problems.

Figure 4 shows the architectural approach of PhoneGap: On the top level is a platform-independent application based on JavaScript which makes use of the interface provided by PhoneGap. Beneath is a platform-dependent JavaScript wrapper library, which maps high-level interface function calls to platform-dependent driver libraries. These native libraries are implemented in the common language of the target platform (Java on Android, Objective C on iOS, etc.) and can easily access sensors and hardware through the provided platform interfaces.

Of course, PhoneGap can only provide a subset of all available functions, but the existing interface should be feasible for most of the applications. Moreover, there are several platform-dependent plugins for PhoneGap (e.g., NFC library) which are available under a free license.

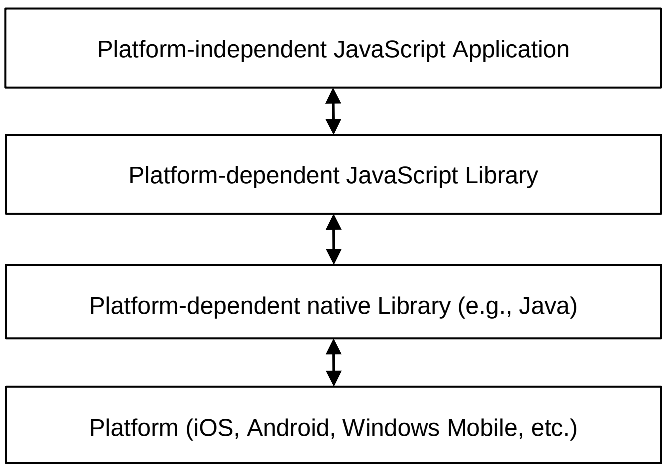

Platform-independent JavaScript Application
Platform-dependent JavaScript Library
Platform-dependent native Library (e.g., Java)
Platform (iOS, Android, Windows Mobile, etc.)

Figure 4: Basic layering concept of PhoneGap.

2.2.2 How to Integrate PhoneGap in an Android Application

You can find a short tutorial of how to integrate PhoneGap into an Android application on the PhoneGap homepage [`http://phonegap.com/start#android`]. As you should have already installed the Android SDK tools and Eclipse plugins, it is sufficient to download the newest version of PhoneGap and follow the instructions on the homepage:

- Create a new folder “www” in the “assets” directory of your current project.
- Create an “index.html” file in the same directory and include the PhoneGap JavaScript library.
- Add the platform-dependent jar library to your project libraries.
- Copy the “xml” folder to the “res” folder of your current project.
- Adapt the Java source code of the main activity as described on the homepage. You can usually find this file in the “src” folder of your project environment.
- Adapt the “AndroidManifest.xml” file according to the homepage. Note that you should remove permissions which you do not need later such that users will not get suspicious of your application.

2.2.3 Creating a PhoneGap Application Using Eclipse Plugin

Open Eclipse and go to “Help → Install New Software…” and enter the following URL `http://svn.codespot.com/a/eclipselabs.org/mobile-web-development-with-phonegap/trunk/download/` to install the PhoneGap Eclipse plugin. It allows you to create Android projects based on PhoneGap and additional JavaScript libraries such as Sencha Touch or jQuery Mobile: Go to “File → New → Other… → PhoneGap Wizards → PhoneGap for Android Project”. Select the preferred version of PhoneGap, decide whether you want to include jQuery Mobile and Sencha Touch to your project and what the project should contain (e.g., a minimal example). Enter a valid name, select the version number of the supported Android API and enter a valid package name. A new Android project will be created, which does not need any further adaptions (library inclusion, adding permissions to manifest file, etc.).

A detailed overview of the complete PhoneGap API may be found on `http://docs.phonegap.com/en/edge/index.html`. Some basic concepts of plugin development and issues on various platforms are given on the wiki of PhoneGap: `http://wiki.phonegap.com/w/page/16494772/FrontPage`.

2.3 Sencha Touch 2

2.3.1 General Features

Like PhoneGap, Sencha Touch 2 is an open-source library. It is only based on JavaScript and optimized for usage in mobile environments such as smartphones or tablets. Its main restriction is that it is only available on WebKit-based browser like Google Chrome or Apple's Safari. Nevertheless, most mobile platforms (Android, iOS, Symbian) provide a WebKit rendering engine such that this requirement is no restriction.

Sencha Touch 2 is a GUI rendering library and allows the programmer to build custom applications with minimal effort. It provides features for accessing external data sources in multiple formats via different interfaces (e.g., JSON data on a RESTful server) and displaying the data in an appropriate way. Moreover, it is possible to add custom HTML and JavaScript code which integrates well with the Sencha Touch application (see Section 2.3.3). In the WP1 prototype, for example, the OpenLayers library is used to display trees on a map. If the user taps on a tree icon, a Sencha Touch component is rendered which displays the detail data of the tree.

Sencha Touch is available under four different licenses (confer `http://www.sencha.com/products/touch/license/`):

1. Free commercial software license: if you do not want to distribute and share the source code with your users.
2. Commercial software license for embedded devices: if your application reaches less than 5.000 devices per year, you do not have pay anything; otherwise you have to contact Sencha.
3. Commercial OEM license: if you want to build a (commercially licensed) SDK or mobile application builder. This might become an interesting option if MDAF will support automatic creation of commercial mobile applications.
4. Open source license: if your software is based on an open-source license compatible with GPLv3, Sencha Touch may be licensed under the conditions of GPLv3.

2.3.2 Modular Creation of Applications

Sencha Touch is only based on JavaScript and CSS, so it is sufficient to provide an empty HTML file which includes the needed libraries and style sheets. The usual document structure of your application might look like this (note that the name of individual files and folders may vary):

- index.html: nearly empty HTML file which only includes JavaScript libraries and CSS files.
- app.js: is the starting point of your application. It is mainly used for development, debugging and testing. If you want to deploy your application, you should prefer to include the app-all.js file.
- app-all.js: contains the complete application in a single JavaScript file. This file is necessary for deployment and is generated by the Sencha build environment as described in Section 2.3.4.
- app.jsb3: configuration file for the Sencha build environment which creates the app-all.js file. On Windows, this file has to be edited manually, on MacOS, it should be created automatically by the Sencha building tools.
- app: folder containing the complete application, split into several modules. Contains the following subfolders:
    - controller: contains the controller specified in the main file (app.js in our case) of your application. Note that the file must have the same name as the specified controller (e.g., the file name of controller 'Main' is 'Main.js', which is located in the app/controller subfolder). A controller is responsible for the control flow and event handling of a view or the complete application.
    - model: contains all used models of your application. A model is a description of the

used data structures and may also contain validators, i.e., checking routines whether specified values are within given boundaries or satisfy given conditions. Like for controllers, the name of the files must correspond to the specified names in the main application file.

- store: contains all used data stores. A store is related to a model such that Sencha Touch knows how to map external data to internal objects. A store can either contain static data specified as JSON objects in the source code or provide an interface to external data sources. The naming convention is the same as for controllers and models.
- view: contains all needed views of the application. There are several types of views which are often related to a data store. E.g., there is a list view which may be used to display a number of values to the user. Views may also be nested, which means that a view may contain another view component. As for all modules of Sencha Touch, the name of the view has to correspond with the name specified in the main application file.

A minimal Sencha Touch 2 application consists of only a few files. In the main application file, which is called app.js within the current documentation, you specify the main configurations of the application:

```
Ext.application({
   name: 'myAppName',
   appFolder: './app',
   controllers: ['myController'],
   models: ['myModel'],
   stores: ['myStore'],
   views: ['myView'],

   launch: function() {
           Ext.Viewport.add({
                   xtype: 'myAppMainPanel'
           });
   }
});
```

The first parameter lets you specify a name of your application. This name is very important for all used modules because their names include the name of the application as prefix (e.g., a view has to be named “myAppName.view.myView”). The second parameter *appFolder* tells Sencha Touch where to look for all involved modules. The default value is “./app” such that this parameter could be omitted for our example. The following four parameters specify the names of the used modules, which are located in the previously described directories. The last function, which is called as soon as the application is ready, adds a new item to the current viewport, i.e., the blank screen of the application. The type of the item is “myAppMainPanel”. This type is the xtype specified in the view “myView” configuration, which is shown below:

```
Ext.define('myAppName.view.myView', {
   extend: 'Ext.Panel',
   xtype: 'myAppMainPanel',
   requires: ['myAppName.view.myPanel'],
   config: {
           ...
   }
});
```

The *Ext.define* function tells Sencha Touch to register a new module with the given name. JavaScript does not provide a clean object-oriented approach, such that inheritance is often implemented differently than it is done in Java or C++. In our case, the current view *extends* the standard panel component (Ext.Panel) of Sencha Touch 2 and registers this newly created type as “myAppMainPanel”. Thus, any other view module can make use of this type by providing the corresponding *xtype*. If the component depends on other customized modules, they can be specified in the *requires* parameter. The *config* parameter is comparable to an object constructor and allows you to specify settings of the current component. It is also possible to associate a store component with a view by adding the following line to the configuration of the view component:

```
store: 'myStore'
```

Thus, the current view component has access to all saved data in the specified store. Several view types such as list and tree views automatically display parts of the given data if a corresponding template is given:

```
itemTpl: '<b>{surname}</b>, {firstname}'
```

This configuration of a list view tells Sencha Touch to display the surname in a bold font weight, followed by the first name in standard weight. Thus, if the store contains multiple items with surname and first name, it will create a list and replace the template specifications, i.e., all text between the curly brackets, by the corresponding values saved in the store component.

Every store needs a model in order that Sencha Touch can identify the field names and the structure of the data. Below, you can find a minimal example, which defines two data fields of type string. The first field is called “surname”, the second “firstname”. The field names have previously been used by the item template of the list view.

```
Ext.define('myAppName.model.myModel', {

   extend: 'Ext.data.Model',
   config: {
           fields: [
                   {name: 'surname', type: 'string'},
                   {name: 'firstname', type: 'string'}
           ]
   }
});
```

Besides data type string, also float, integer and Boolean types are available. A store may easily be implemented if the data is known in advance and does not need to be retrieved from an external server:

```
Ext.define('myAppName.store.myStore', {
   extend: 'Ext.data.Store',

   config: {
           model: 'myAppName.model.myModel',
           data: [
                   {firstname: 'John', surname: 'Doe'},
                   {firstname: 'Donald', surname: 'Duck'}
           ]
   }
});
```

In case the data is saved in an external database and is available in a supported format (e.g., JSON or XML), the store does not contain the actual data, but a proxy configuration including the address of the server and the expected data format:

```
Ext.define('myAppName.store.myStore', {
   extend: 'Ext.data.Store',

   config: {
           model: 'myAppName.model.myModel',
           defaultRootProperty: 'data',
           proxy: {
                   type: 'jsonp',
                   url: 'http://www.example.com/db/',
                   reader: {
                           type: 'json',
                           rootProperty: 'data'
                   }
           },
           autoLoad: true
   }
});
```

The last line tells Sencha Touch that the data shall initially be loaded per default. Otherwise, you have to trigger the retrieving manually by calling the load method of the store.

Finally, we need a controller which is responsible for all *actions* that shall be done on specified *events*. Below is a minimal example which shows a welcoming message to the user if the application is started:

```
Ext.define('myAppName.controller.myController', {
   extend: 'Ext.app.Controller',
   config: {
           refs: {
                   main: 'myAppMainPanel'
           },
           control: {
                   'myAppMainPanel' : {
                           initialize: 'showWelcome'
                   }
           }
   },
   showWelcome: function(component, object) {
           alert('Welcome ' + username + '!');
   }
});
```

The *control* field contains a single JSON object including definitions of event listeners for all components. The components are usually referenced by their *xtype* (which is “myAppMainPanel” in our case), but more complex combinations are possible, e.g., if the application contains multiple components of the same *xytype* but with different ids. In this case, a filter function like “'button[itemId="myButton"]'” may be specified. The events and the corresponding handlers are part of each control definition: In our example, every component of type “myAppMainPanel” will call the “showWelcome” function when it is initialized. As we only have one component of this type, which is initialized when the application starts, the “showWelcome” function is only called once. The function has to be an object of the current controller and must not be defined as a global function. Nevertheless, it is also possible to implement the handler function immediately after the event. The parameters of the handler function depend on the type of the event and are documented in the *events* section of each component in the Sencha Touch API documentation. Note that it does not matter whether you define multiple controllers, e.g., one per view component, or a single one for the whole application.

The current Section gave a short overview of the modular programming approach of Sencha Touch 2. A nice video tutorial of how to display a set of data in a list view by using the MCV paradigm is available on `http://www.sencha.com/learn/meet-the-list-component`. The complete online documentation of Sencha Touch 2 is available on `http://docs.sencha.com/touch/2-0/`.

2.3.3 Integration of External Libraries and Individual Functions into Sencha Touch Applications

So far, it has only been described how to create a small application completely based on the functionality and features provided by Sencha Touch 2. What if you need components or functions which are not part of the Sencha Touch 2 library? An easy way is to set up some global variables, which exchange data between a Sencha Touch function and an externally implemented JavaScript function. Thus, it is possible to run a background function which gets the current location using the interface of PhoneGap and saves the coordinates to a global object. In a Sencha Touch controller function, you can simply read out the saved values of the global variable.

Turning now to a more complex scenario: In nearly all mobile prototypes of ENVIROFI, some kind of map view is needed to display observations or weather phenomena on a map. Although there is a so called map component in Sencha Touch 2, it only provides the basic interface of the Google Maps API and needs a permanent Internet connection. Thus, displaying cached map data is not easily possible. When using OpenLayers (`http://www.openlayers.org/`), you are free to use any map data and displaying observations may be implemented easily. Though, OpenLayers needs the id of a HTML div layer to render a map. One possibility to implement a map rendered in OpenLayers within your Sencha Touch application is to create a nearly empty component view:

```
Ext.define('myAppName.view.MapView', {
   extend: 'Ext.Component',
   xtype: 'myMap',
   config: {
           title: 'Map View',
   }
});
```

If you add the following lines to your controller, you can easily call a custom function which initializes the map view:

```
...
   control: {
...
           'vtltreemap': {
                   show: 'showMap'
           },
...
   },
   showMap: function(component, object) {
           showMap(component.getId());
   },
...
```

The function “showMap” might be implemented in any included JavaScript file and expects the id of the HTML div element as the single input parameter. A minimum implementation of the function just creates a new OpenStreetmap view in the layer with the given id:

```
function showMap(mapID) {
   var map = new OpenLayers.Map({
           div: mapID,
           projection: "EPSG:900913",
           displayProjection: "EPSG:4326",
           numZoomLevels: 18
   });
   // create OSM layers
   var mapnik = new OpenLayers.Layer.OSM();
   map.addLayer(mapnik);
}
```

For a more detailed description of the OpenLayers library take a look at the documentation [`http://docs.openlayers.org/library/index.html`].

We now showed a simple example how an external function may be called from Sencha Touch. How does the other way work, i.e., how can you access components and data structures of Sencha Touch from an external JavaScript function? This is in fact easier than you might expect because Sencha Touch provides quite powerful functions to access all elements which are part of the application:

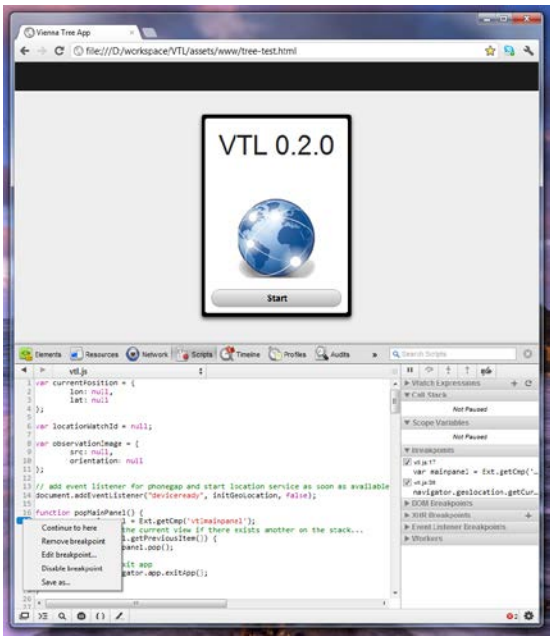
- By using “Ext.getComp(String id)” you can get a component with the specified id. This function is globally available and may be used by any of your JavaScript functions. You now have access to all methods of the component and may change the associated store component or the html text to be displayed.
- By using “Ext.data.StoreManager.lookup(String id)” you can get a store module with the specified id. You can now add or remove data, change the URL of the associated proxy object or reload data from the specified server.

There are some even more powerful features of the Sencha Touch component and storage managers, which allow you to get all child components of a container or with a specified CSS class, etc. These are described in the documentation of the “Ext.DomQuery” object [`http://docs.sencha.com/touch/2-0/#!/api/Ext.DomQuery`].

2.3.4 Testing and Debugging

It is recommendable to use a browser for debugging the general functionality of your application and a smartphone, e.g., Samsung Galaxy Nexus or Samsung Nexus S, to verify features including hardware. Using the emulator is only advisable if you do not possess any real hardware because the testing process is more difficult and some events may only be created correctly on real hardware.

If you are developing on Windows or Linux, the best browser for debugging is Google Chrome. Like for the “FireBug” plugin of Mozilla Firefox, you have the possibility to inspect created HTML elements in the source code or set break points and watch expressions in your JavaScript sources. Just open the root html file with Google Chrome and adapt the window size such that the proportions are similar to the display of your device. Figure 5 shows the debugging view of Google Chrome which can be enabled by pressing “<Ctrl>+<Shift>+I” or by clicking right on an element and selecting “Inspect Element”.



```
var currentPosition = {
        lon: null,
        lat: null
};

var locationWatchId = null;

var observationImage = {
        src: null,
        orientation: null
};

// add event listener for phonegap and start location service as soon as available
document.addEventListener("deviceready", initGeoLocation, false);

function popMainPanel() {
            l = Ext.getCmp('vtlmainpanel');
           the current view if there exists another on the stack...
           l.getPreviousItem()) {
           panel.pop();

           xit app
           gator.app.exitApp();
```

Figure 5: A Sencha Touch Application displayed in Google Chrome. At the top, you can see the rendered HTML output, below is the debug view, which can be activated by going to the main menu (right upper corner) and selecting “Tools” → “Development Tools”.

For debugging, you should include the debug version of Sencha Touch 2 and other used libraries. Moreover, do not pack and obfuscate your own application (e.g., by using the Sencha building tools) for debugging. The debug versions of the libraries are not obfuscated and provide more information in case an exception is thrown. Thus, you can find the reason why a call to the API does not work properly more easily.

For finding errors, you can also make use of the console log which is part of Google Chrome and the Android browser: Just add

```
console.log("My logging message")
```

to your JavaScript code and you will find the output message either in the console tab of the development tools of Google Chrome or in the logcat output of Eclipse in case you debug on an emulator or a device. Other available logging functions are described on `http://getfirebug.com/logging`.

2.3.5 Packaging a Sencha Touch 2 Application

A quite undocumented feature of Sencha Touch 2 is its building environment, which is currently available for MacOS, Windows and Linux. As it is still a beta version, it does not provide all documented features, but is good enough for packaging. Download the current version for your operating system on `http://www.sencha.com/products/sdk-tools/download/` and install it. Make sure that you register the following directories in your path environment variable (on Windows, this can be done by clicking right on “MyComputer” → “Properties” → “Advanced System Settings” → “Environment variables”):

- <SENCHA-SDK-INSTALL-DIR>
- <SENCHA-SDK-INSTALL-DIR>/jsbuilder
- <SENCHA-SDK-INSTALL-DIR>/command

Start a new terminal or command line (“cmd.exe” on Windows) and enter “sencha build”. You should now get a help output of the Sencha Touch SDK tool. Go to the root directory of your application and create a new file called “app.jsb3”. A minimal example contains the following lines:

```
{
   "projectname": "myProjectName",
   "licenseText": "Copyright(c) 2012 myself",
   "builds": [{
           "name": "myApp",
           "target": "app-all.js",
           "compress": true,
           "files": [
                   {
                           "path": "app/controller/",
                           "name": "myController.js"
                   },
...
           ]
   }]
}
```

You may specify the name of the project, a short copyright information and the configurations of each application. The application has a name, an output file name (“app-all.js” in our case), an option that the output file should contain compressed JavaScript code and a listing of all needed files. Note that the *path* parameter of a file has to end with a slash “/”, otherwise, the file will not be included in the output file. Moreover, the order of file including is important because you run into problems if dependencies (the *required* parameter of components) cannot be resolved. Thus, make sure that you first include files which do not depend on other components. You can invoke the packaging process by simply typing

```
sencha build -p app.jsb3 -d .
```

in your terminal. This tells the building tools of Sencha Touch to create your application by using the configuration given in “app.jsb3” and using the current directory “.” as destination folder. You should now see a new file “app-all.js” in the current directory. If you want to deploy your package or test it on a mobile device, make sure you include this file instead of the “app.js” in your root html file.

You can find even more information about the Sencha Touch building tools on the Robert Dougan’s blog [`http://robertdougan.com/posts/packaging-your-sencha-touch-2-application-using-the-sencha-sdk-tools`].

2.4 OpenLayers

tbd

2.5 CouchDB

2.5.1 General Features

CouchDB is an Apache project of a document-based database implemented in Erlang. There are releases for Windows, MacOS and Linux as well as iOS and Android. CouchDB provides powerful synchronization features, a rudimentary user management and an extension for geospatial queries (see Section 2.5.3). In contrast to relational (SQL) databases, documents of the same category (entity or class) do not need to have the same attributes. All data is saved as key-value-pair in a JSON object. CouchDB provides a RESTful interface, meaning that all operations (insert, edit or delete item) can be done via different HTTP methods like PUT, GET or DELETE. As Sencha Touch 2 provides a controller for RESTful services and all objects are saved in JSON format, it is quite easy to interact with a CouchDB database.

2.5.2 Views and List Views

Unlike for typical SQL databases, you cannot define a simple select-from-where statement which retrieves you information from one or multiple tables. However, CouchDB provides a mechanism based on two functions, which are called “map” and “reduce”. The “map” function is responsible for filtering, whereas the “reduce” function implements aggregate functions like calculating the number of rows, finding the maximum value of the result set etc.

The standard output format of a view is JSON, meaning that the result set consists of an array of JSON objects. If you need other output formats, e.g., XML or CSV, you can implement list views: List views are based on views and allow you to send any type of strings to the user while having access to the objects and data fields of the view.

2.5.3 GeoCouch

tbd

3 Application Deployment

3.1.1 Creation of Android Installable Packages (APK Files)

Creating an installable Android application is quite easy if you make use of the Android Eclipse plugins: Just click right on the Android project in the explorer view and select “Android Tools” → “Export Signed Application Package…”. Although it is also possible to build unsigned packages, these can only be installed via the Android debugging tools and is thus not recommendable if you want to deploy the application. If you want to build a signed package for the first time, you have to create a new keystore. Otherwise, you can just use keys from an existing keystore. Provide a location and enter a password for the keystore. Press next and create a new key or select an existing one. Note that the passwords for the keystore and the key are usually not the same! If you create a new key, you have to specify at least one of the certificate fields, i.e., your name, company, etc. Finally, select a valid destination and safe the file.

The file ending of an Android application is “apk”. However, the file type is just a zip archive containing the compiled code of your application. Thus, your HTML and JavaScript code is visible to all users who have access to your apk-File. Moreover, when downloading the application, some browsers (e.g., Microsoft Internet Explorer) rename the file to the specified file type such that the ending is “zip”. If you want to install the app on your Android device, you first have to recover the original file ending.

3.1.2 Deploying on Google Play (former Android Market)

tbd

4 Conclusions

tbd

References

- Android SDK and Eclipse Tools:
  - `http://developer.android.com/sdk/index.html`
  - `http://developer.android.com/sdk/win-usb.html`
  - `http://www.eclipse.org/downloads/`
  - `http://developer.android.com/sdk/eclipse-adt.html`
- PhoneGap:
  - `http://phonegap.com/`
  - `http://phonegap.com/start#android`
  - `http://svn.codespot.com/a/eclipselabs.org/mobile-web-development-with-phonegap/trunk/download/`
  - `http://docs.phonegap.com/en/edge/index.html`
  - `http://wiki.phonegap.com/w/page/16494772/FrontPage`
- Sencha Touch 2:
  - `http://www.sencha.com/products/touch/`
  - `http://www.sencha.com/products/touch/license/`
  - `http://www.sencha.com/learn/meet-the-list-component`
  - `http://www.sencha.com/learn/intro-to-layouts/`
  - `http://docs.sencha.com/touch/2-0/`
  - `http://robertdougan.com/posts/packaging-your-sencha-touch-2-application-using-the-sencha-sdk-tools`
  - `http://www.sencha.com/products/sdk-tools/download/`
- OpenLayers
- CouchDB
  - `http://guide.couchdb.org/`
  - `http://wiki.apache.org/couchdb/`