convertigo
MASHUP YOUR BUSINESS

Getting started with Convertigo Mobilizer

First Sencha-based project tutorial

CEMS 6.0.0

TABLE OF CONTENTS

Convertigo Mobilizer overview 1

Introducing Convertigo Mobilizer 1-1

Convertigo Mobilizer purpose 1-1

Convertigo Mobilizer architecture 1-2

My first Convertigo Mobilizer Sencha-based project 2

Prerequisites 2-1

Key project 2-1

Convertigo Mobilizer project 2-2

Project description 2-2

Opening the sample project from the Studio 2-2

Setting up the project 2-13

Creating a project basis and setting mobile features 2-13

Creating a project basis 2-14

Cleaning the basic project 2-19

Creating the mobile user interface 2-24

Mobile environment presentation 2-24

CSS directory 2-25

IMG directory 2-26

JS directory 2-27

SOURCES directory 2-28
INDEX.HTML file 2-34
CONFIG.XML file 2-35
Creating the mobile user interface 2-37
Testing the application 2-45
Test Platform presentation 2-45
Convertigo toolbar 2-47
Project's elements 2-47
Execution result panel 2-47
Executing the application from the Test Platform 2-50
Integrating dynamic data to the application 2-54
Preparing the target Convertigo Web Service 2-55
Integrating a list with a call to Convertigo and its response 2-59
Executing the application from the Test Platform 2-67
Improving the application functionalities 2-71
Displaying a details sheet on list items disclosure 2-71
Setting up the details sheet user interface 2-75
Integrating a details sheet with a call to Convertigo and its response 2-84
Use the application from your own mobile device 2-88
Deploying projects in a Convertigo Cloud server 2-89
Case of a Convertigo Studio testing user 2-89
Case of a Convertigo Cloud client 2-92
Running mobile web application 2-96
Building and installing native mobile application 2-97

LIST OF FIGURES

Figure 1 - 1: Convertigo Mobilizer - Functional architecture 1-2

Figure 1 - 2: Convertigo Mobilizer - Sencha-based project's technical architecture 1-3

Figure 2 - 1: Launching the New Project wizard 2-3

Figure 2 - 2: Opening sample project in New project wizard 2-3

Figure 2 - 3: Selecting Web Integration sample project 2-4

Figure 2 - 4: CWI sample project in Projects view 2-5

Figure 2 - 5: Launching the New Project wizard 2-6

Figure 2 - 6: Opening sample project in New project wizard 2-6

Figure 2 - 7: Selecting Mobilizer Sencha-based sample project 2-7

Figure 2 - 8: Selecting Sencha Touch framework 2-8

Figure 2 - 9: Downloading Sencha Touch framework 2-9

Figure 2 - 10: Saving ZIP file 2-10

Figure 2 - 11: Selecting Sencha Touch framework 2-11

Figure 2 - 12: Sample project creation confirmation message 2-11

Figure 2 - 13: CWI and CMob sample projects in Projects view 2-12

Figure 2 - 14: CWI and CMob sample projects connector editors 2-13

Figure 2 - 15: New project wizard - Selecting a project type 2-14

Figure 2 - 16: New project wizard - Setting a project name 2-15

Figure 2 - 17: New project wizard - Selecting target devices 2-16

Figure 2 - 18: New project wizard - Configuring mobile features and look and feel 2-17

Figure 2 - 19: New project wizard - Selecting Sencha Touch framework 2-18

Figure 2 - 20: Project creation confirmation message 2-19
Figure 2 - 21: sampleMobilizerSencha project appearing in the Projects view 2-19
Figure 2 - 22: Deleting unused sequence 2-20
Figure 2 - 23: Deletion confirmation message 2-21
Figure 2 - 24: Cleaned Mobilizer project 2-21
Figure 2 - 25: Project folder in Project Explorer 2-22
Figure 2 - 26: Deleting unused resources 2-22
Figure 2 - 27: Deletion confirmation message 2-23
Figure 2 - 28: Cleaned project resources 2-23
Figure 2 - 29: Project folder in Project Explorer 2-24
Figure 2 - 30: DisplayObjects folder and mobile folder and subfolders 2-25
Figure 2 - 31: Project look and feels style sheets 2-26
Figure 2 - 32: Images in `img` directory 2-27
Figure 2 - 33: JavaScript library files 2-28
Figure 2 - 34: Sources of the mobile application pages 2-29
Figure 2 - 35: Main JavaScript file app.js in editor 2-30
Figure 2 - 36: Main app.js: app global structures and variables 2-31
Figure 2 - 37: Main app.js: app global variables and usefull functions 2-32
Figure 2 - 38: Main app.js: app startup properties definitions 2-33
Figure 2 - 39: Links to application resources in index.html 2-35
Figure 2 - 40: Mobile application general description in config.xml 2-36
Figure 2 - 41: Mobile application required accesses and root in config.xml 2-37
Figure 2 - 42: Default searchForm.js file displayed in editor - part 1 2-39
Figure 2 - 43: Default searchForm.js file displayed in editor - part 2 2-40
Figure 2 - 44: Updating searchForm.js file in editor 2-41
Figure 2 - 45: Non-updated search button in searchForm.js file 2-42
Figure 2 - 46: Edited searchForm.js file 2-42
Figure 2 - 47: Updating search button handler function 2-43
Figure 2 - 48: Edited searchForm.js file 2-43

Figure 2 - 49: Default topToolbar.js file displayed in editor 2-44
Figure 2 - 50: Updating topToolbar.js file in editor 2-44
Figure 2 - 51: Convertigo test platform URL 2-46
Figure 2 - 52: Test platform for the sampleMobilizerSencha project 2-46
Figure 2 - 53: NewConnector and default_transaction with no variable in test platform 2-47
Figure 2 - 54: Empty Execution result panel in test platform 2-47
Figure 2 - 55: Android mobile device in test platform 2-48
Figure 2 - 56: Build mobile applications panel in test platform 2-49
Figure 2 - 57: Built application for Android device in test platform 2-50
Figure 2 - 58: Application running in Android mobile device in test platform 2-51
Figure 2 - 59: Changing form values in application search form 2-52
Figure 2 - 60: Error popup after clicking Search button 2-53
Figure 2 - 61: Application running in Web browser in test platform 2-54
Figure 2 - 62: Opening CWI connector in editor 2-55
Figure 2 - 63: CWI connector editor (DOM Tree, Web Browser, XPath Evaluator) 2-56
Figure 2 - 64: Available test cases of searchGoogleWithLimit transaction 2-57
Figure 2 - 65: Running test case in Convertigo Studio 2-58
Figure 2 - 66: Transaction result displayed in Output tab of Connector editor 2-59
Figure 2 - 67: Default listDisplay.js file displayed in editor - part 1 2-61
Figure 2 - 68: Default listDisplay.js file displayed in editor - part 2 2-62
Figure 2 - 69: Updating listDisplay.js file in editor - part 1 2-63
Figure 2 - 70: Edited listDisplay.js file 2-64
Figure 2 - 71: Updating listDisplay.js file in editor - part 2 2-64
Figure 2 - 72: Edited listDisplay.js file 2-65
Figure 2 - 73: Updating listDisplay.js file in editor - part 3 2-65
Figure 2 - 74: Updating listDisplay.js file in editor - part 4 2-66
Figure 2 - 75: Edited listDisplay.js file 2-67
Figure 2 - 76: Activation of loading mask after clicking Search button 2-68
Figure 2 - 77: Result list after search perfomed in iPhone4 device in test platform 2-69

Figure 2 - 78: Item selected in result list in test platform 2-70
Figure 2 - 79: Default behavior on list item activation in listDisplay.js file 2-72
Figure 2 - 80: Updating list item activation behavior in listDisplay.js file 2-72
Figure 2 - 81: Edited listDisplay.js file 2-73
Figure 2 - 82: Result list after search perfomed in iPhone4 device in test platform 2-74
Figure 2 - 83: Details sheet in iPhone4 device in test platform 2-75
Figure 2 - 84: Default details.js file displayed in editor - part 1 2-77
Figure 2 - 85: Default details.js file displayed in editor - part 2 2-78
Figure 2 - 86: Updating details.js file in editor - part 1 2-79
Figure 2 - 87: Updating details.js file in editor - part 2 2-80
Figure 2 - 88: Edited details.js file 2-80
Figure 2 - 89: Updating details.js file in editor - part 3 2-81
Figure 2 - 90: Edited details.js file 2-81
Figure 2 - 91: Updating details.js file in editor - part 4 2-82
Figure 2 - 92: Edited details.js file 2-82
Figure 2 - 93: Details sheet in Android device in test platform 2-83
Figure 2 - 94: Accessing target web site in test platform 2-84
Figure 2 - 95: Updating details.js file in editor - part 5 2-85
Figure 2 - 96: Edited details.js file 2-86
Figure 2 - 97: Activation of loading mask on details sheet 2-87
Figure 2 - 98: Details sheet after image search perfomed in test platform 2-88
Figure 2 - 99: Deploying a Convertigo project 2-89
Figure 2 - 100: Email with Convertigo Shared Cloud registration certificate 2-90
Figure 2 - 101: Registrating Convertigo Shared Cloud in Studio 2-91
Figure 2 - 102: Deploy window automatically filled 2-92
Figure 2 - 103: Empty Deploy window 2-93
Figure 2 - 104: Filling fields in deploy window 2-94
Figure 2 - 105: Progression of project deployment 2-95
Figure 2 - 106: Project deployment confirmation message 2-95

Figure 2 - 107: Flashing mobile web application QR code with device 2-97
Figure 2 - 108: Convertigo Studio administration URL 2-98
Figure 2 - 109: Convertigo Cloud administration URL 2-98
Figure 2 - 110: Convertigo Server administration authentication page 2-99
Figure 2 - 111: Convertigo Server administration Home page 2-100
Figure 2 - 112: Accessing Configuration page 2-101
Figure 2 - 113: Expending Mobile Builder section in Configuration page 2-102
Figure 2 - 114: Updating PhoneGap build account 2-102
Figure 2 - 115: Updated configuration confirmation message 2-103
Figure 2 - 116: Android mobile device on test platform on Convertigo Cloud server 2-104
Figure 2 - 117: Build mobile applications panel on Convertigo Cloud server 2-104
Figure 2 - 118: Editing Build application endpoint to Convertigo Cloud URL 2-105
Figure 2 - 119: Applications build in progress on PhoneGap 2-105
Figure 2 - 120: Applications build finished 2-106

1 Convertigo Mobilizer overview

This chapter presents the purpose, concepts and architecture of **Convertigo Mobilizer (CMob)**, one of the modules of Convertigo Enterprise Mashup Server (C-EMS) platform.

- Introducing Convertigo Mobilizer
- Convertigo Mobilizer purpose
- Convertigo Mobilizer architecture

1.1 Introducing Convertigo Mobilizer

Convertigo Mobilizer is a mobile enterprise application platform (*MEAP*) included in C-EMS that allows to easily create cross-platform mobile applications over existing business applications and other enterprise data sources. **Mobilizer** is a module of C-EMS platform that extends other modules to design, develop and build mobile UIs.

Convertigo Mobilizer encompasses a large number of capabilities:

- *reusing* any existing application from a company as is - no need to rewrite any piece of code,
- *accessing* instantaneously any resource, even resources not providing any APIs,
- *combining* and *reusing* existing business logic from one or several applications,
- *collecting* data extracted from these applications and data sources and *mashup* these data,
- *creating* at once web apps and multi-platform mobile applications (for iPhone, iPad, Android, and BlackBerry6),
- *building* and *publishing* native mobile applications on target application stores.

1.2 Convertigo Mobilizer purpose

The purpose of **Convertigo Mobilizer (CMob)** is to create a mobile web application as well as a native mobile application based upon existing business applications and data sources in the company, such as:

- Web Services,
- RSS syndication feed,
- Legacy screens (BULL, IBM),

- HTML pages,
- SQL databases.

Figure 1 - 1: Convertigo Mobilizer - Functional architecture

Using the existing modules of C-EMS (such as **Convertigo Web Integrator**, **Convertigo Legacy Integrator**, **Convertigo Mashup Sequencer**, etc.), Convertigo Mobilizer collects data from existing applications, databases or web services, that are used to feed the mobile application.

*Any type of **Convertigo** project can be considered as a data provider in the context of a **Mobilizer** project.*

Convertigo Mobilizer module allows developping the mobile user interface using standard frameworks, such as:

- ***Sencha Touch*** (JavaScript Object Model),
- ***JQuery Mobile*** (HTML Markup + DOM manipulations),
- ***PhoneGap build*** (native apps build + SQL Lite support).

1.3 Convertigo Mobilizer architecture

Convertigo Mobilizer module's specific functionality is to create the mobile user interface of the application. This section presents the technical architectures of **Convertigo Mobilizer** in the context of a ***Sencha***-based project.

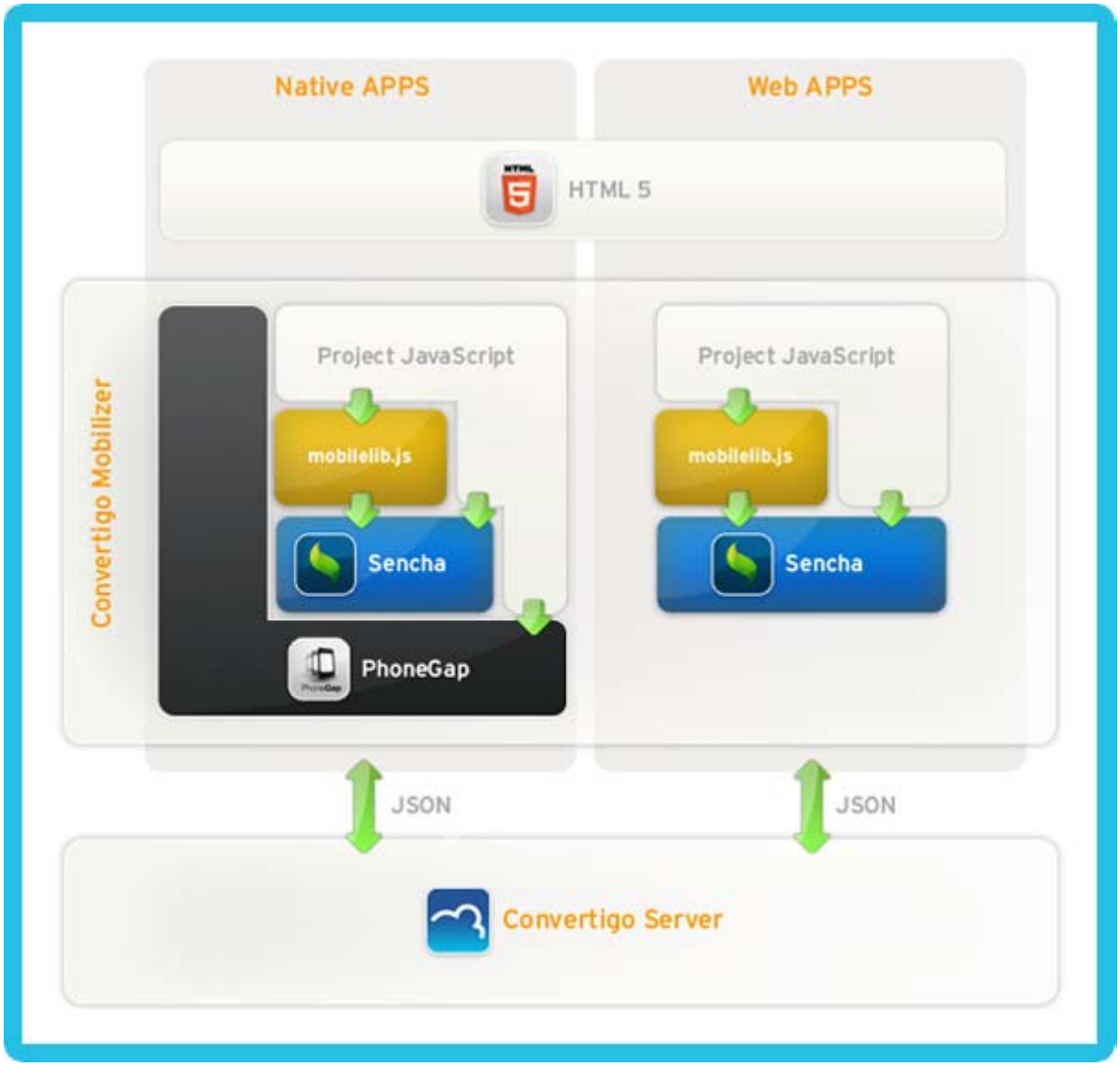


Figure 1 - 2: Convertigo Mobilizer - Sencha-based project's technical architecture

Convertigo Mobilizer allows creating mobile applications presenting data provided by other modules of a **Convertigo Server**. Convertigo Mobilizer interacts with applications and data sources through the Convertigo Server using JSON data structures.

Convertigo Mobilizer produces *mobile web applications* or *native mobile applications*.

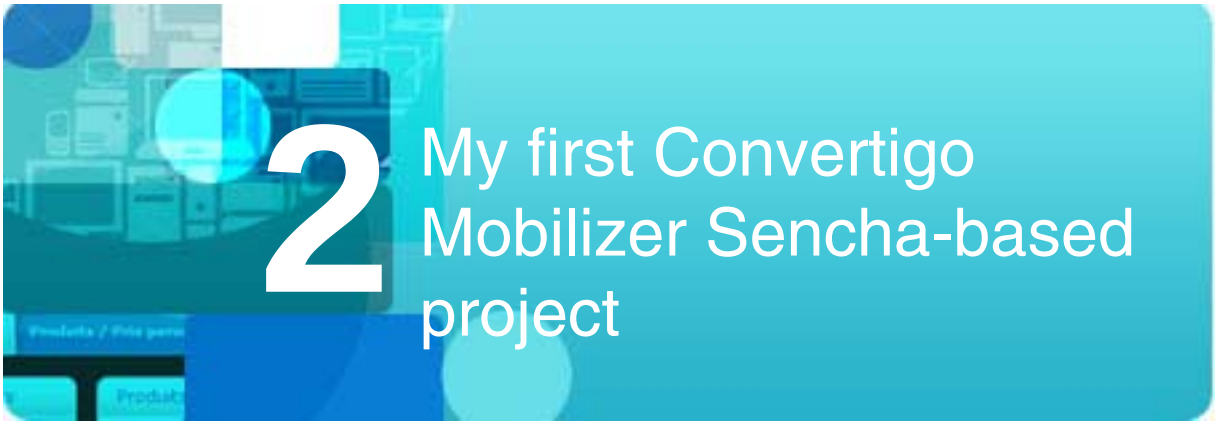
Both are based on ***Sencha Touch*** framework, a mobile JavaScript framework for developing HTML5 web apps that look and feel native, combined with a Convertigo JavaScript library, named "*mobilelib*".

Using this framework, each Convertigo Mobilizer project contains local JavaScript files, also named "*project features*", that are edited by Convertigo developer to create his own mobile user interface.

For native applications, the project is build using ***PhoneGap build***, an HTML5 application platform that allows to author native applications with web technologies and get access to mobile devices's APIs and application stores.

In both cases, the final application is providing high-end HTML5 mobile user experience applications.

*For more information about **Sencha Touch** and **PhoneGap build**, see their respective web sites: http://www.sencha.com/products/touch/ and http://www.phonegap.com/.*



2 My first Convertigo Mobilizer Sencha-based project

This chapter describes the project prerequisites, then gives instructions on how to set up a simple **Convertigo Mobilizer *Sencha***-based project.

- Prerequisites
- Convertigo Mobilizer project
- Setting up the project
- Use the application from your own mobile device

2.1 Prerequisites

Before starting with the creation of the Convertigo Mobilizer Sencha-based project, please read carefully the following sub-section:

- Key project

2.1.1 Key project

This Mobilizer Sencha-based project (called `sampleDocumentationMobilizerSencha`) is based on the **Convertigo Web Integrator** (CWI) `sample_documentation_CWI` project, developed in the context of the *"Getting Started with Convertigo Web Integrator - First project tutorial"* documentation. This project's purpose is to launch a keyword-based Google search and generate an XML output containing Web page titles and URLs returned by Google.

In the context of this tutorial, two new transactions were created in `sample_documentation_CWI` project.

- The first one, named `searchGoogleWithLimit` is similar to the `searchGoogle` transaction developped in the CWI *First project tutorial*, with an extra variable, called `maxPages`, setting the number of result pages to accumulate on Google.
- The other transaction, named `searchGoogleImage`, has for purpose to perform a Google Images search and extract the first image's URL.

These transactions will be used by this Mobilizer project.

It is highly recommended that you review CWI project before starting with this Mobilizer project tutorial.
For more information, refer to the "Getting Started with Convertigo Web Integrator - First project tutorial".

2.2 Convertigo Mobilizer project

This section describes the Mobilizer project you are going to create and how to open the completed sample project from the Studio:

- Project description
- Opening the sample project from the Studio

2.2.1 Project description

The purpose of this **Convertigo Mobilizer (CMob)** project, called `sampleMobilizerSencha`, is to create a mobile application, based on ***Sencha Touch*** framework, that allows to:

- *search* a keyword in Google,
- *display a list* of results,
- *open details* of a list item and display a related image,
- and finally access target web page.

This mobile application should use the CWI `sample_documentation_CWI` project as a REST web service performing the research and providing result data.

In the end, the created mobile web application should be built for specific devices in order to generate native mobile applications to be deployed directly on the smartphones and application stores.

2.2.2 Opening the sample project from the Studio

The completed sample project is stored in Convertigo installation folder.

The Mobilizer project calls transactions defined in the Convertigo Web Integrator sample project `sample_documentation_CWI`. We will therefore open this sample project before opening the sample CMob project.

The following procedure describes how to access it from the Studio using the **New Project** wizard.

To open a sample project from the Studio

1. If not already opened, run the **Convertigo Studio**;
2. In the **Studio** menu bar, select ***File > New > Project*** or click on in the toolbar then select ***Project***.

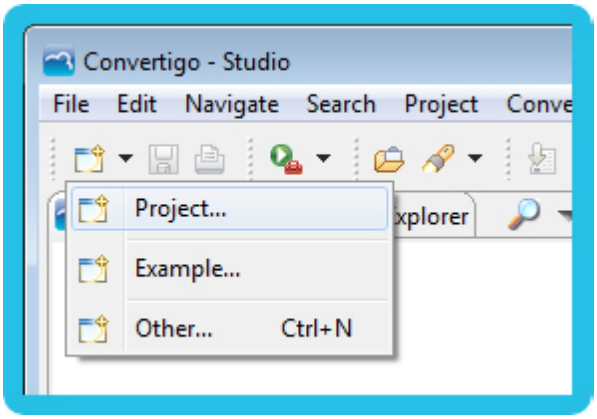


Figure 2 - 1: Launching the New Project wizard

A **New Project** wizard opens.

3 Expand **Convertigo Projects**, then **Samples**.

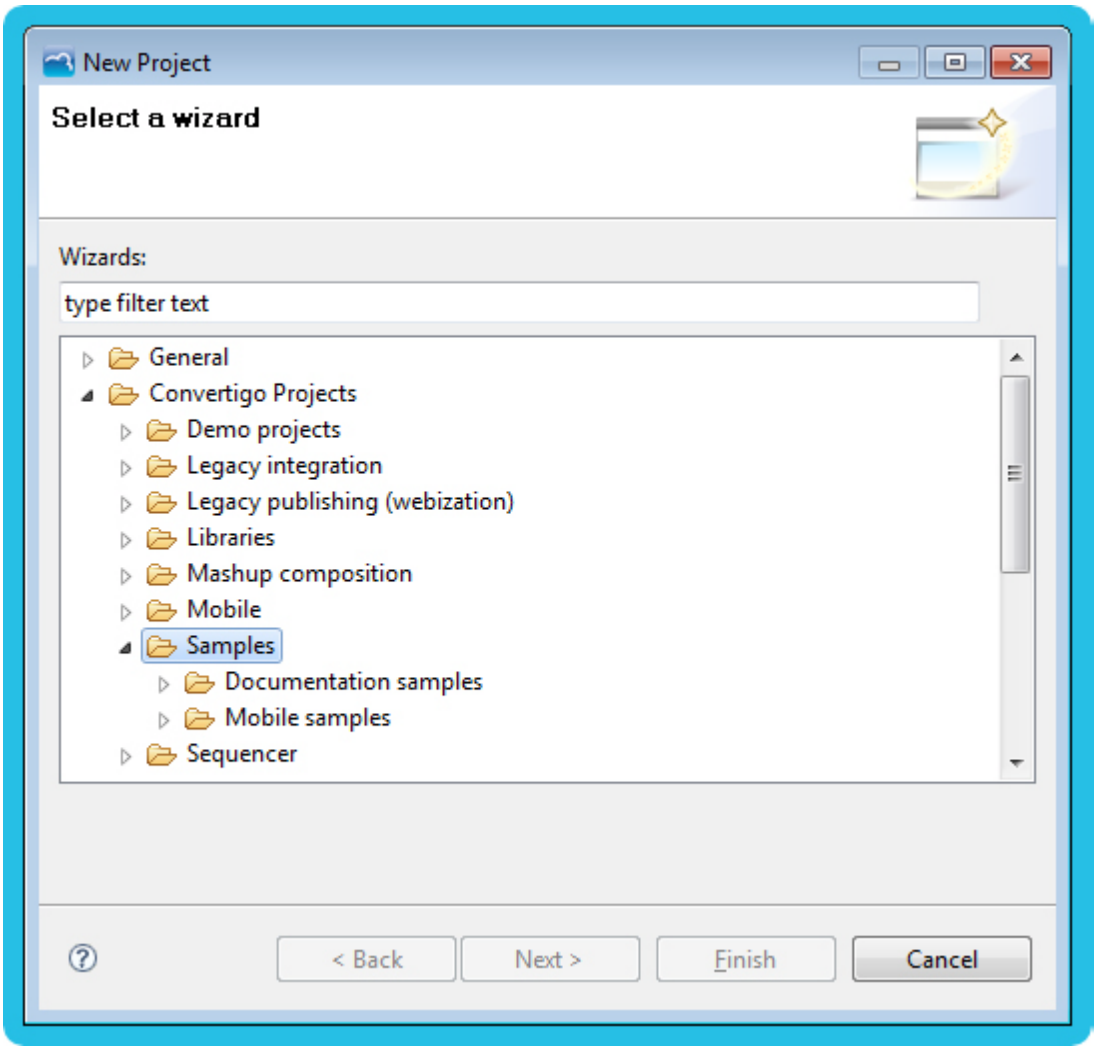


Figure 2 - 2: Opening sample project in New project wizard

4 Expand **Documentation samples** and select the sample project to open (in our case, **Web integration** project):

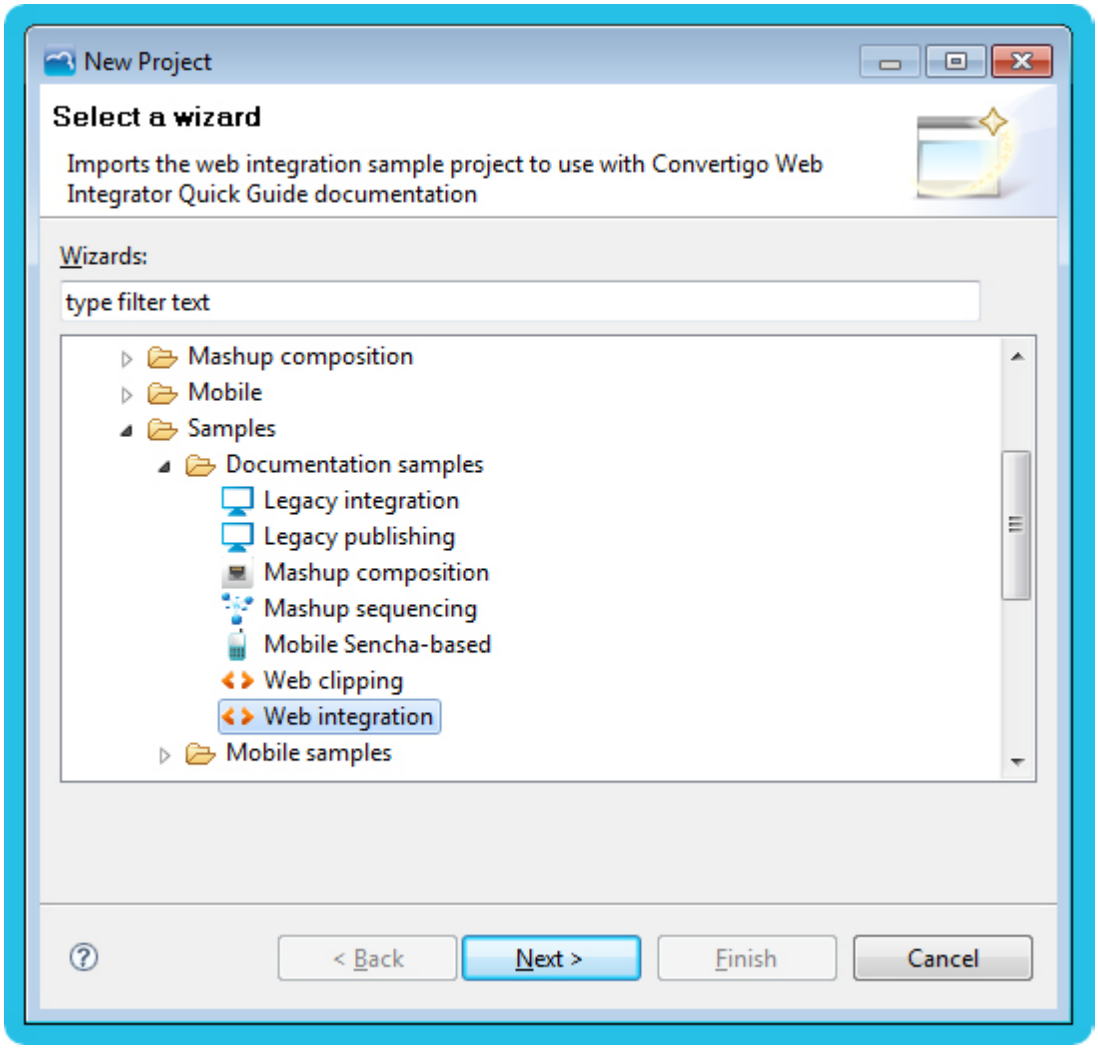


Figure 2 - 3: Selecting Web Integration sample project

5 Click on **Next**, then **Finish**.

The sample project opens in the Studio, you can see its objects in the **Projects** view:

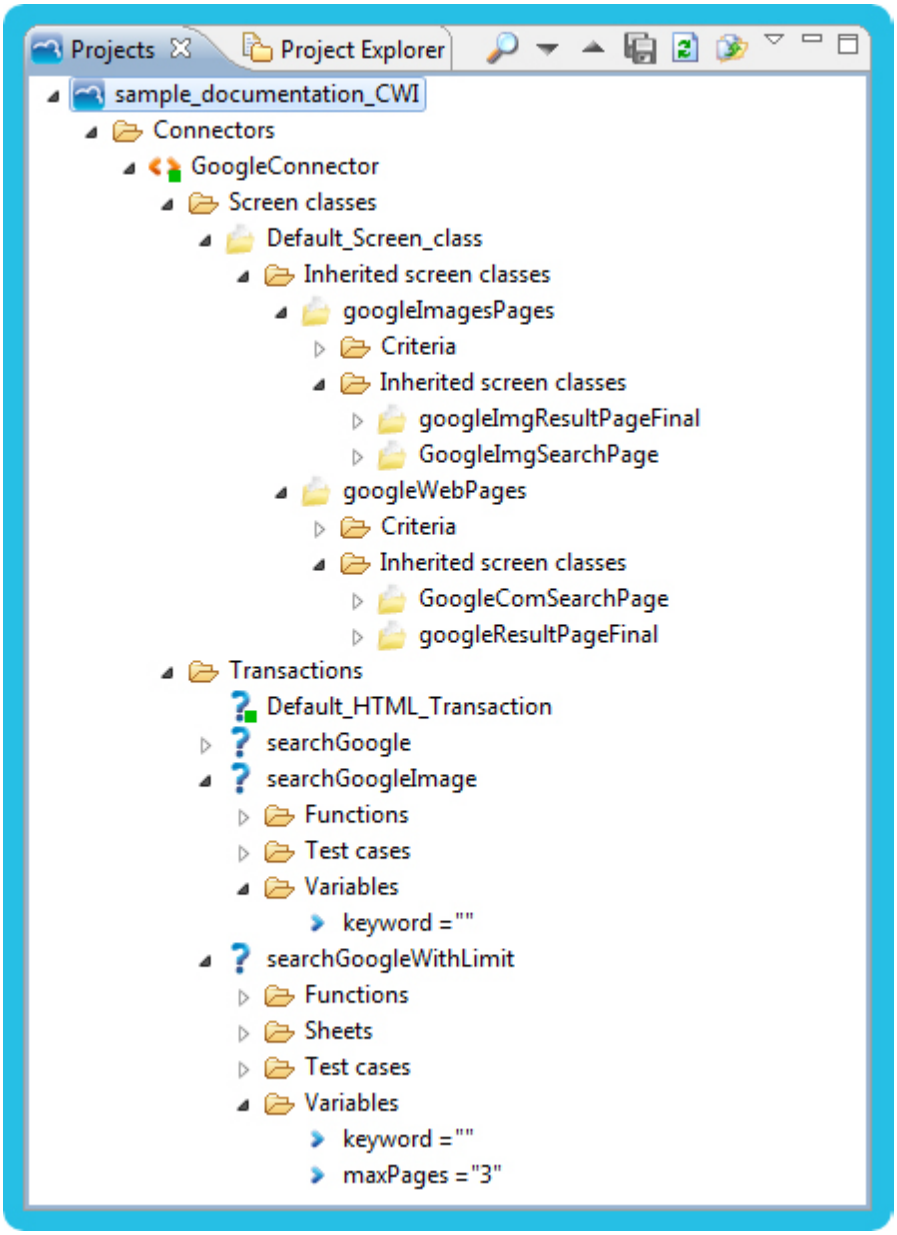


Figure 2 - 4: CWI sample project in Projects view

Now, we can open the sample CMob project. The procedure is a little different for Mobilizer projects.

To open a Mobilizer sample project from the Studio

1 In the **Studio** menu bar, select ***File > New > Project*** or click on in the toolbar then select ***Project***.

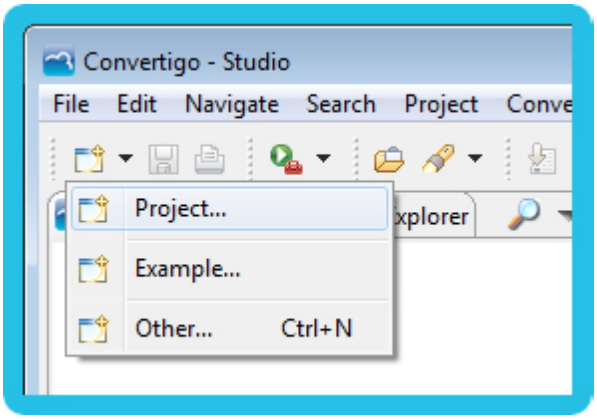


Figure 2 - 5: Launching the New Project wizard

A **New Project** wizard opens.

2 Expand **Convertigo Projects**, then **Samples**.

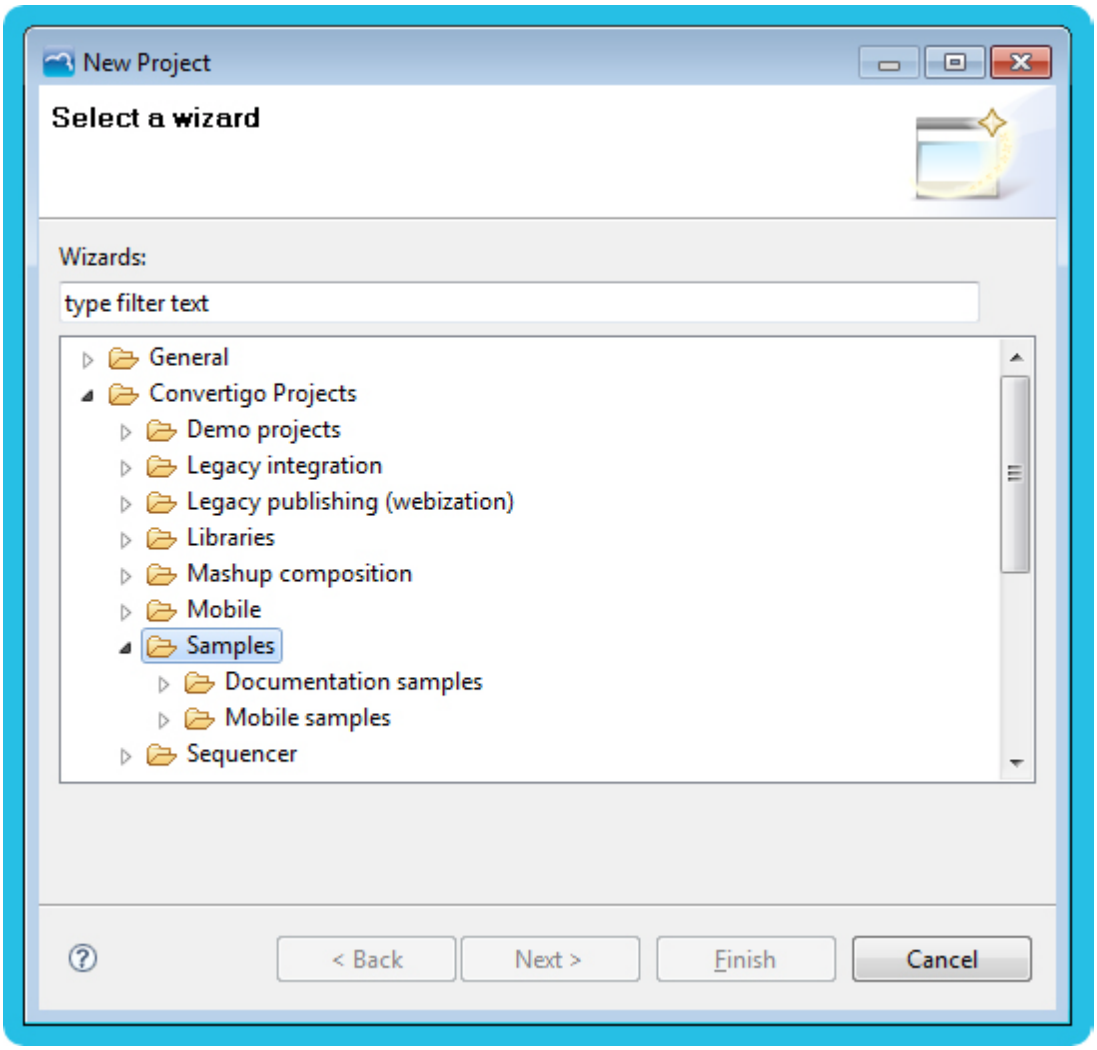


Figure 2 - 6: Opening sample project in New project wizard

3 Expand **Documentation samples** and select the sample project to open (in this case, **Mobile Sencha-based** project):

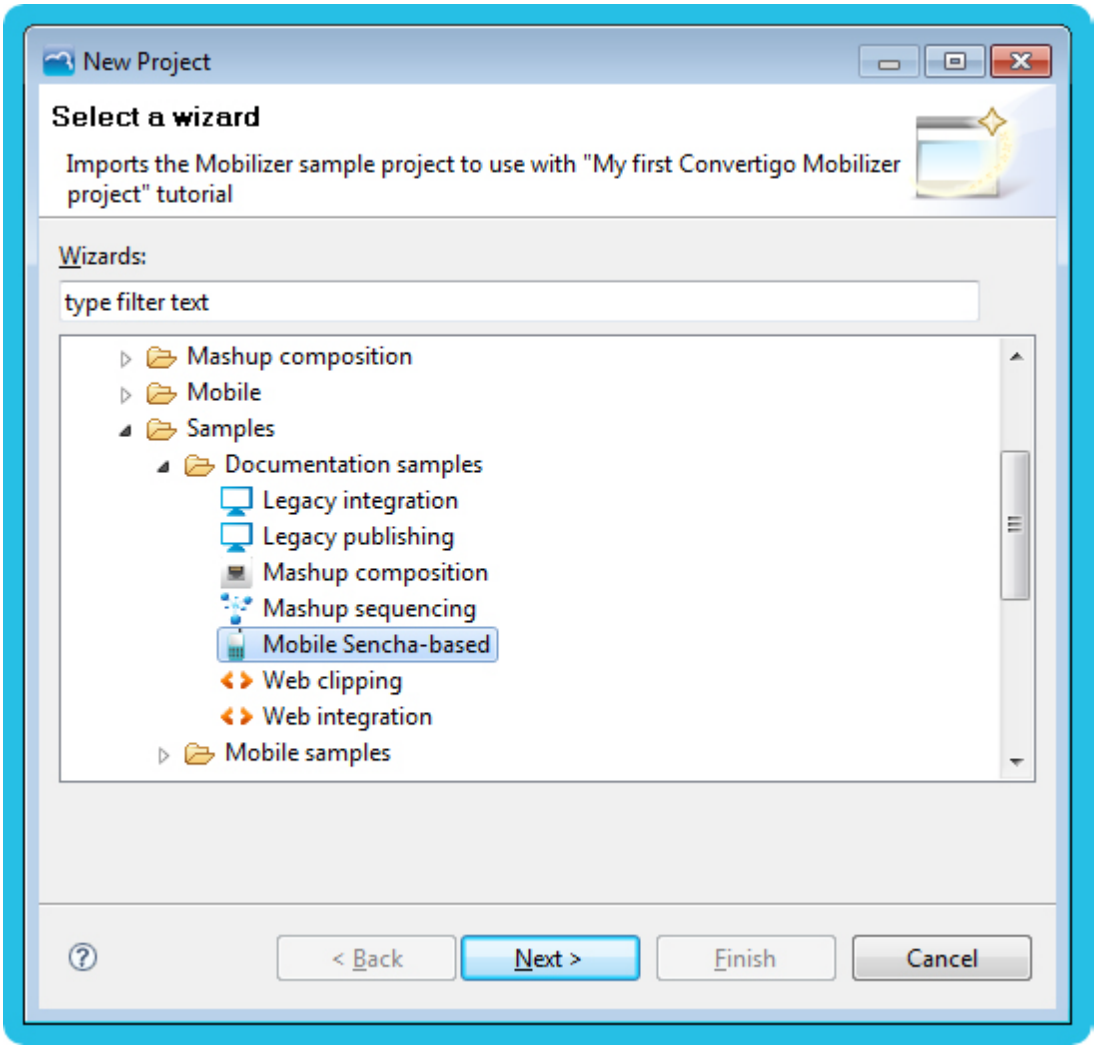


Figure 2 - 7: Selecting Mobilizer Sencha-based sample project

4 Click on **Next**, then confirm the chosen project type by clicking again on **Next**.

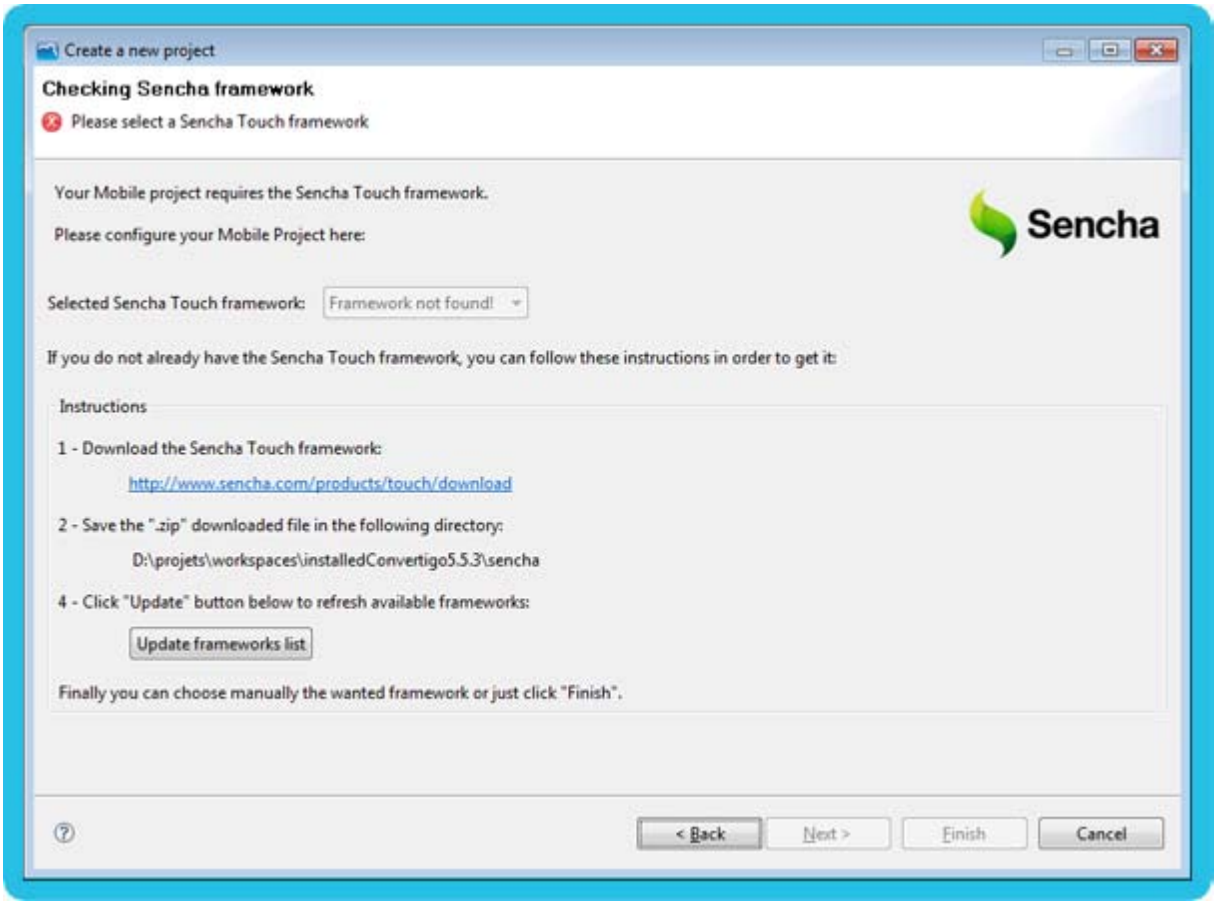


Figure 2 - 8: Selecting Sencha Touch framework

5 In this step, if no ***Sencha Touch*** framework is found in your workspace, you need to download it. To do so, follow the steps 6 to 8 of the procedure.

If a ***Sencha Touch*** framework is found in your workspace, select it and continue with step 9 of the procedure.

6 Download the ***Sencha Touch*** framework at `http://www.sencha.com/products/touch/download`. Access quickly to this page by clicking on the link in the wizard.

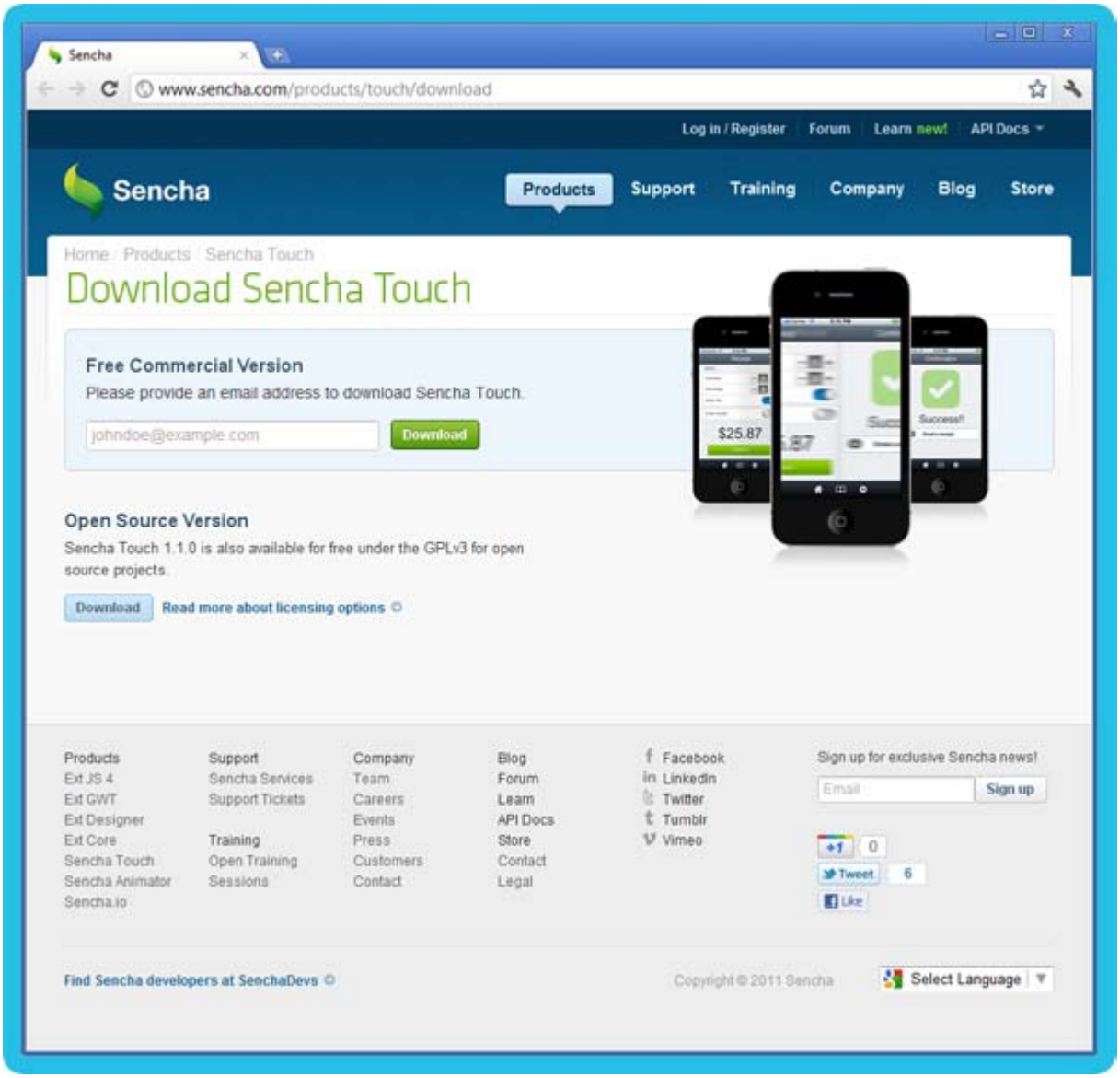


Figure 2 - 9: Downloading Sencha Touch framework

7 Save the "`.zip`" downloaded file in the directory specified in the wizard. This path is automatically calculated to your Convertigo workspace set when installing Convertigo Studio.

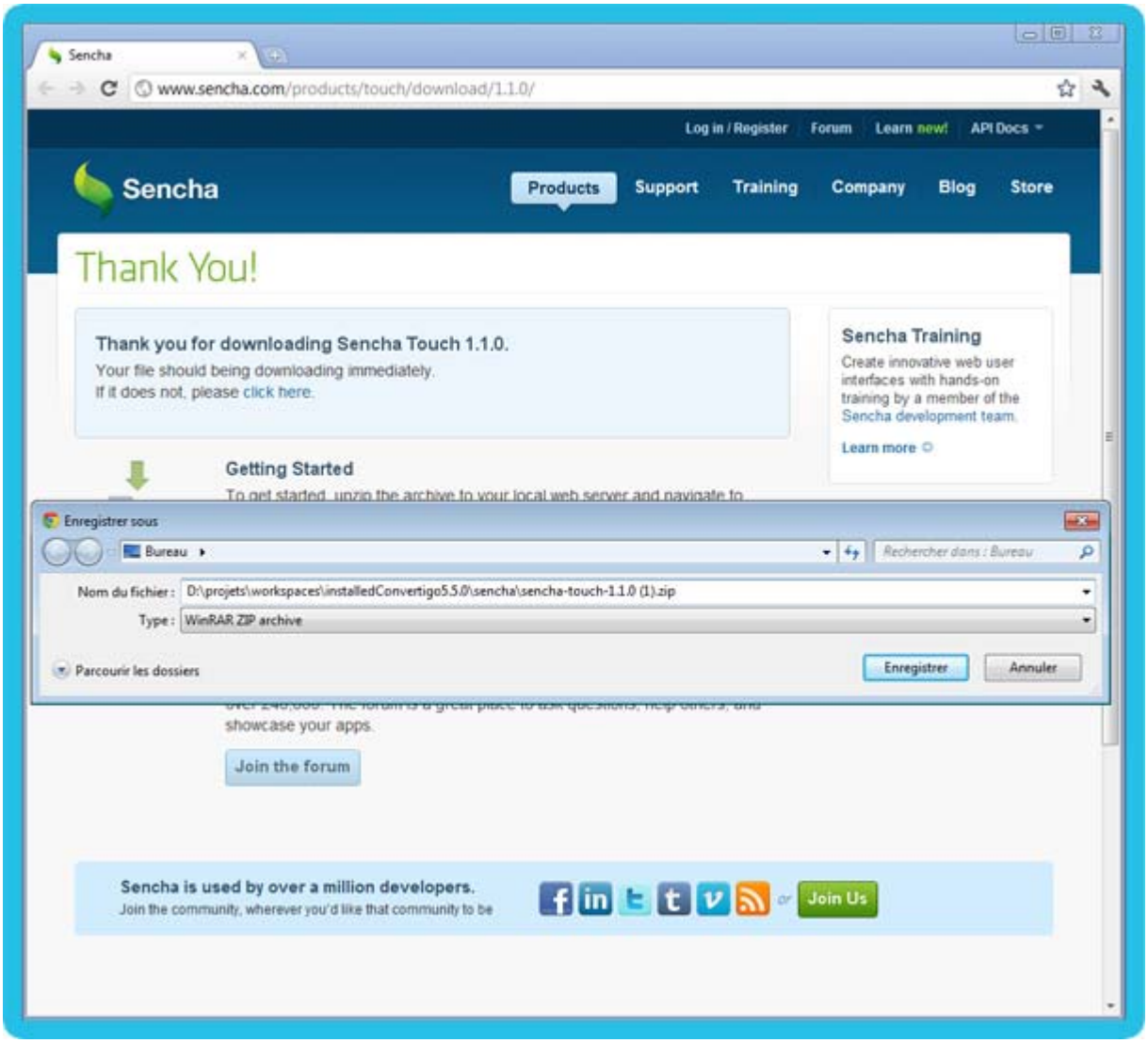


Figure 2 - 10: Saving ZIP file

8 Back on the Convertigo wizard, click on the **Update frameworks list** button to refresh the list of available frameworks. The downloaded framework appears and is selected by default.

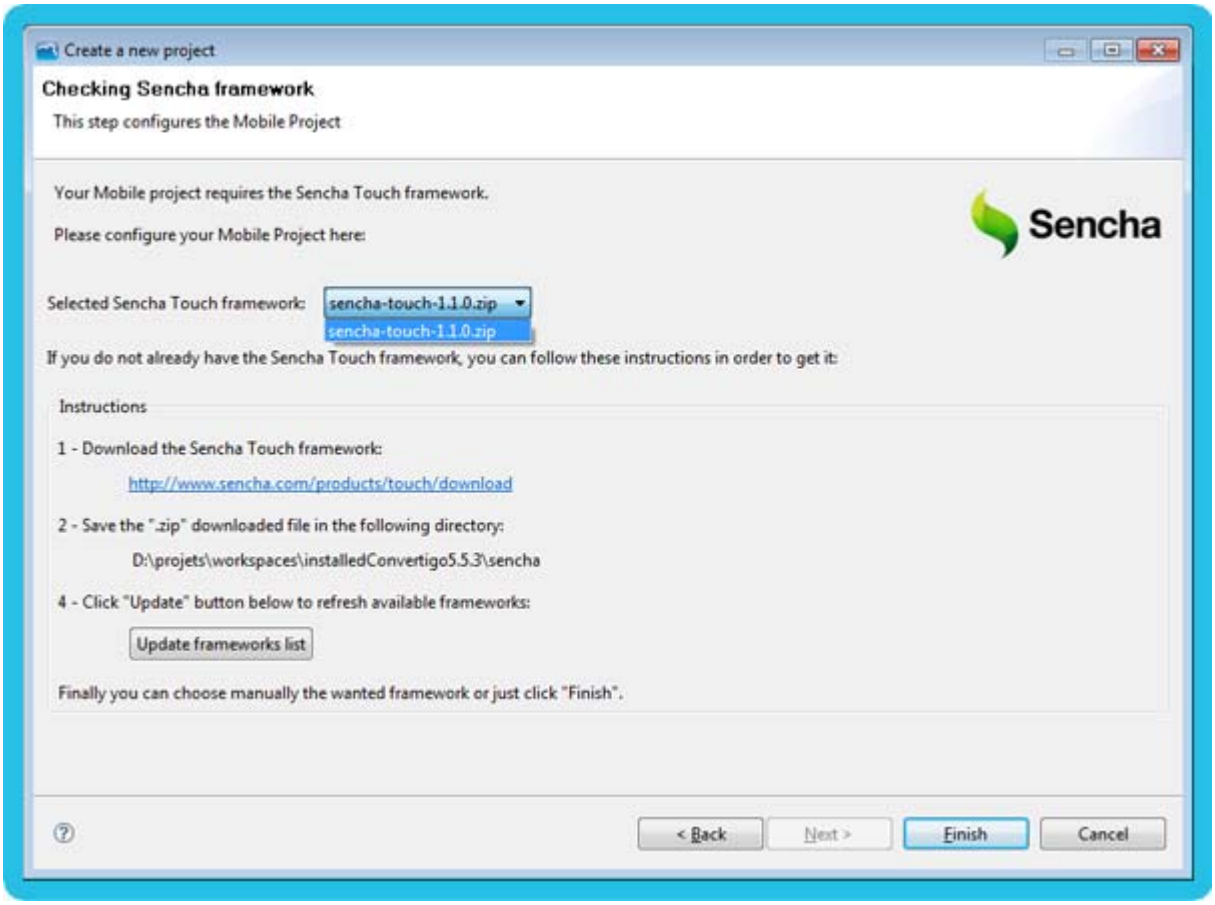


Figure 2 - 11: Selecting Sencha Touch framework

9 Click on **Finish**.

The sample project is loaded in the Studio and a confirmation message is displayed, with a link to reach the test platform of the project.

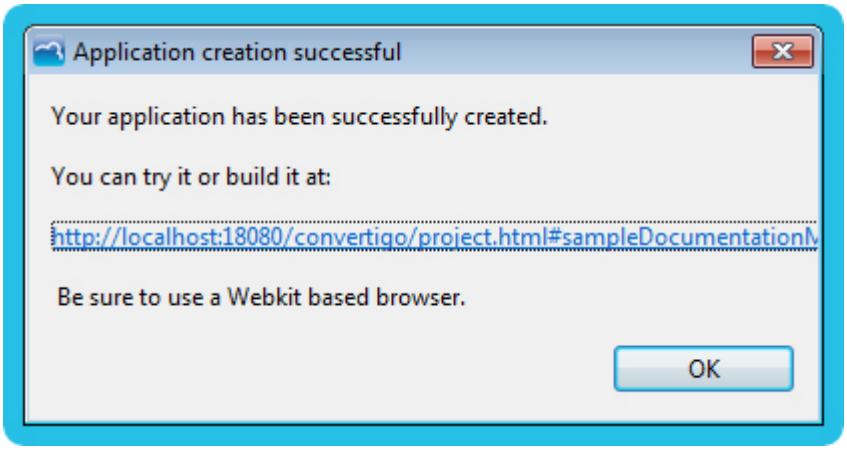


Figure 2 - 12: Sample project creation confirmation message

10 Click on **OK** to close the message.

After the Mobilizer sample project has been opened, both projects appear in the **Projects** view:

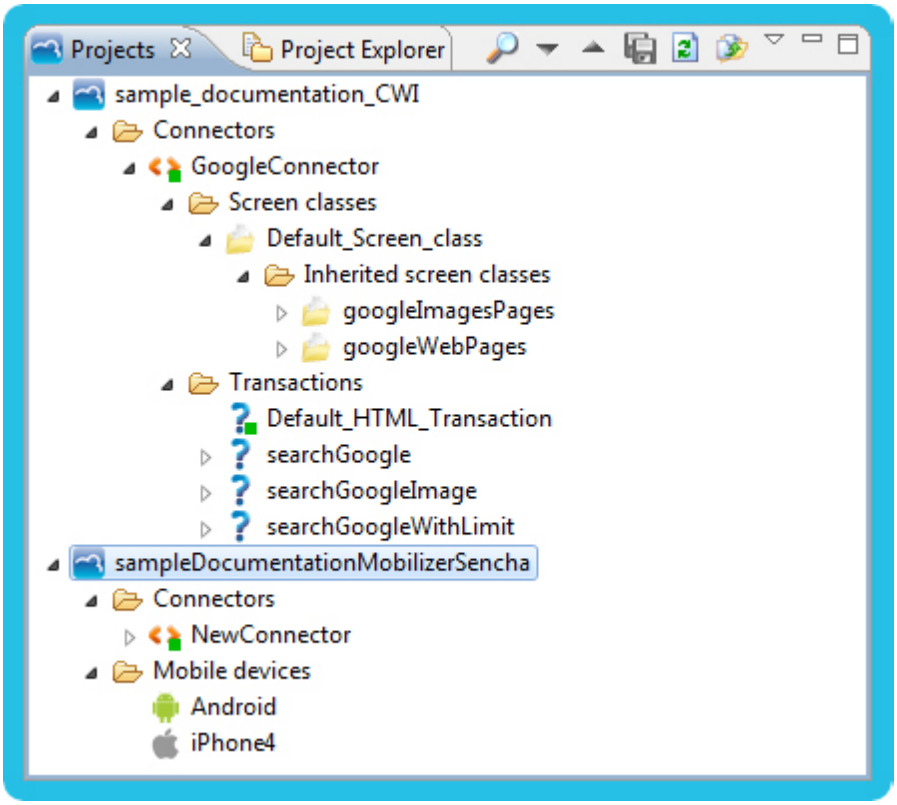


Figure 2 - 13: CWI and CMob sample projects in Projects view

Both corresponding connectors are opened in their editors:

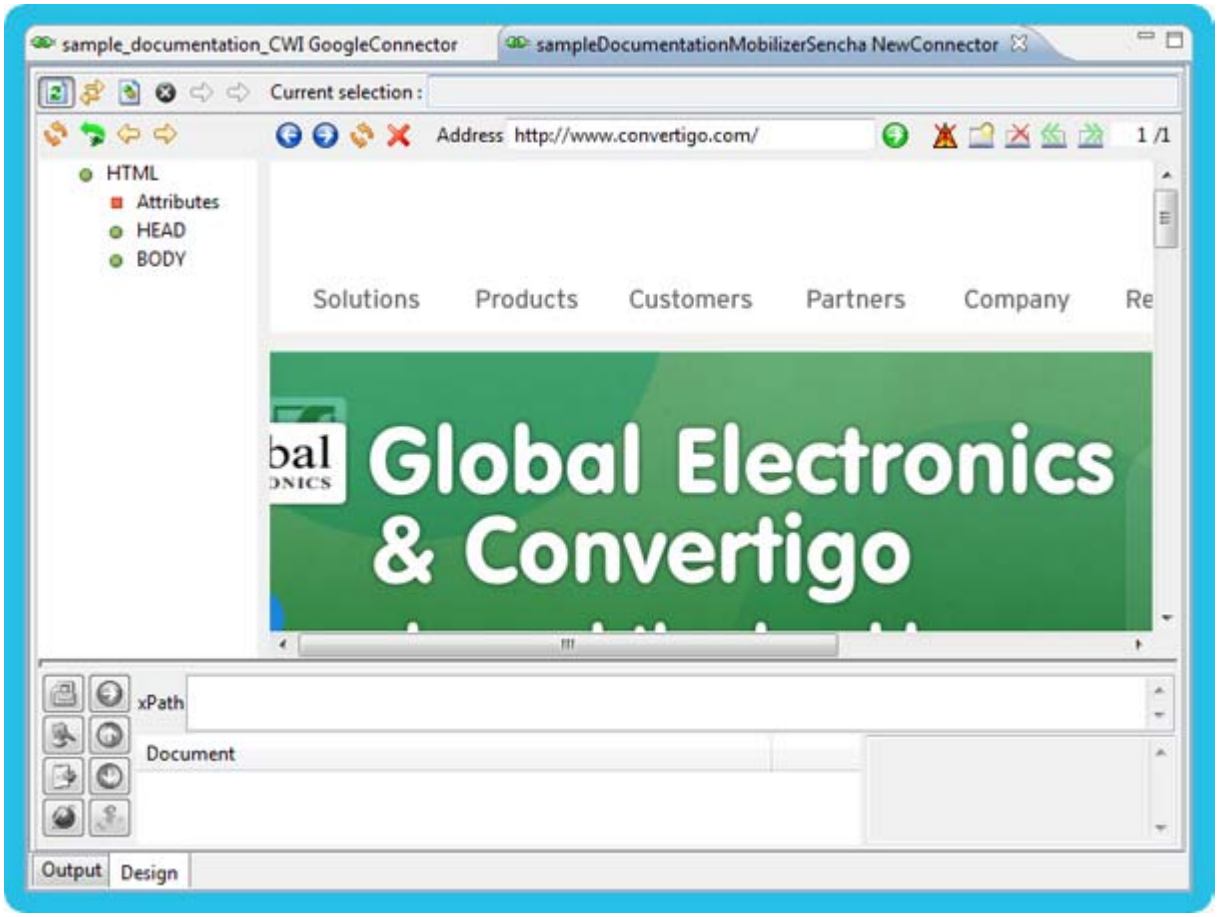


Figure 2 - 14: CWI and CMob sample projects connector editors

As the connector of the Mobilizer project is not used in the mobile application, you can close its editor by clicking on button.

You can now start setting up your own CMob project, `sampleMobilizerSencha`.

2.3 Setting up the project

The following sections explain step by step how to set up your first Convertigo Mobilizer project to create the previously described mobile application:

- Creating a project basis and setting mobile features
- Creating the mobile user interface
- Testing the application
- Integrating dynamic data to the application
- Improving the application functionalities

2.3.1 Creating a project basis and setting mobile features

This section explains how to create a Mobilizer project, choose and set its mobile features (preconfigured panels in the mobile application) and prepare a clean project basis to use in next section, in which the project will be customized.

- Creating a project basis

- Cleaning the basic project

CREATING A PROJECT BASIS

The first step is to create a new CMob project. In this example, the project is called `sampleMobilizerSencha`.

To create a project and set mobile features

1 In the Studio menu bar, select ***File > New > Project*** or click on in the toolbar then select ***Project***.

A **New Project** wizard opens.

2 Expand **Convertigo Projects**, then **Mobile** and select **Sencha/PhoneGap based Project**:

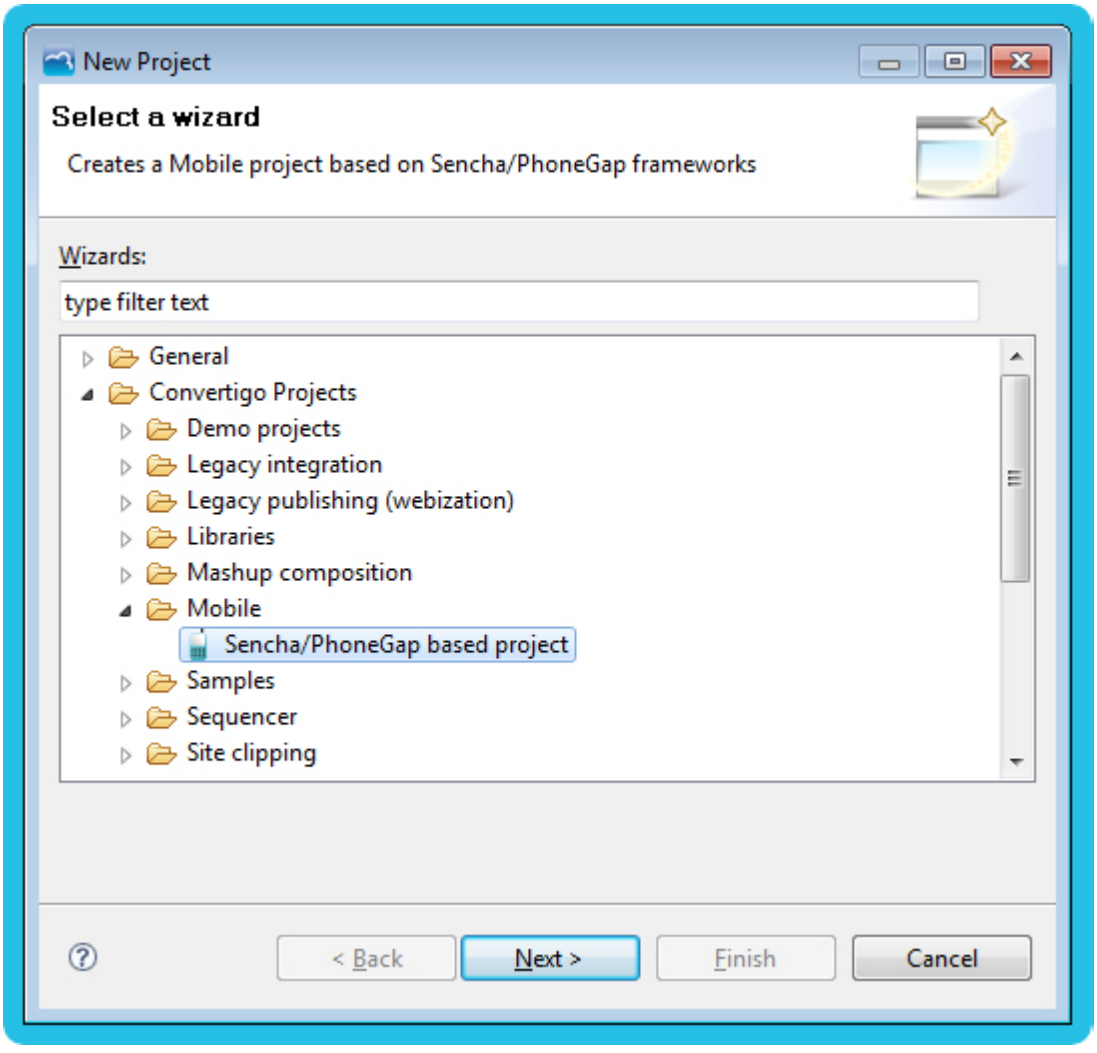


Figure 2 - 15: New project wizard - Selecting a project type

3 Click on **Next**.

4 In the **Project's name** field, type in the project name (for example: `sampleMobilizerSencha`).

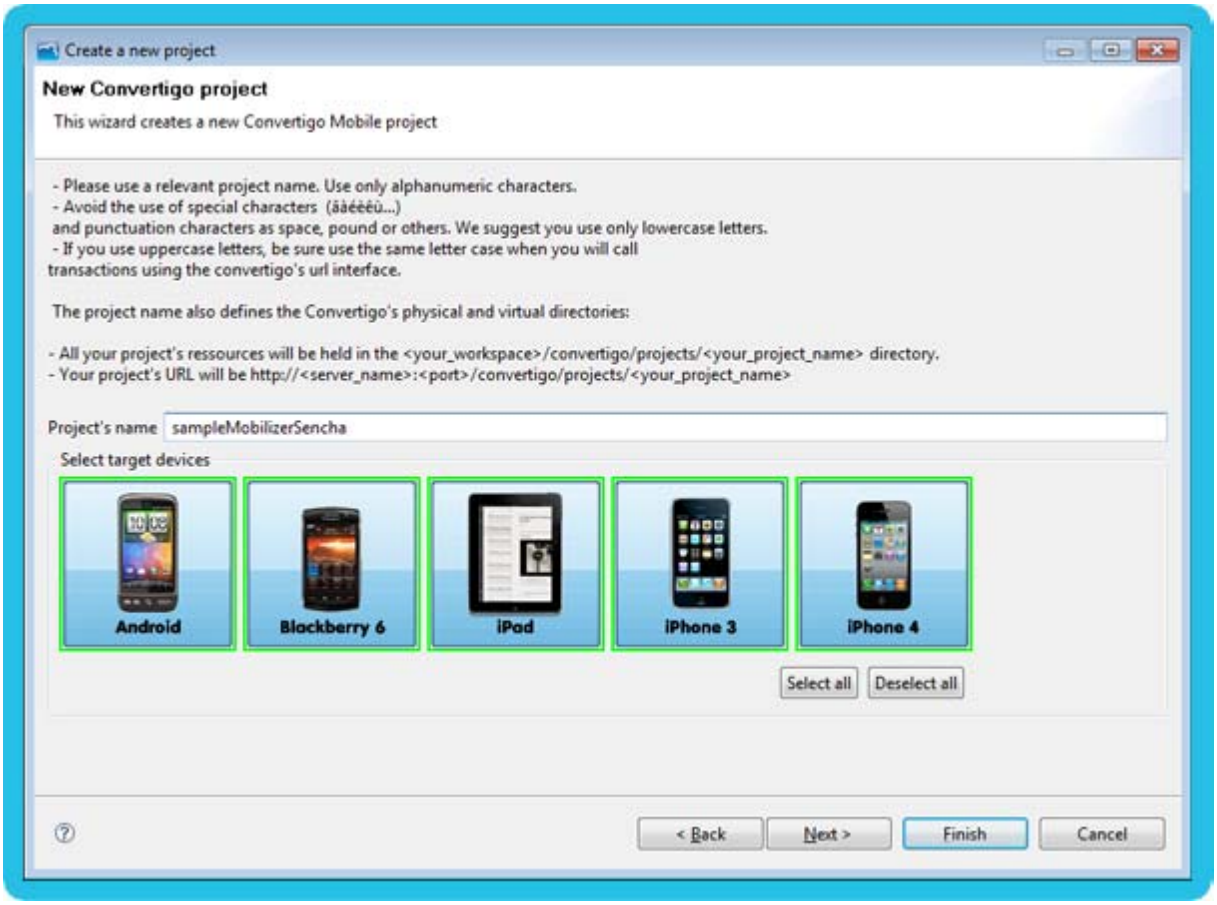


Figure 2 - 16: New project wizard - Setting a project name

5 Then select target mobile devices on which your application should run (you can select one or many target devices). Select for example `Android` and `iPhone 4`:

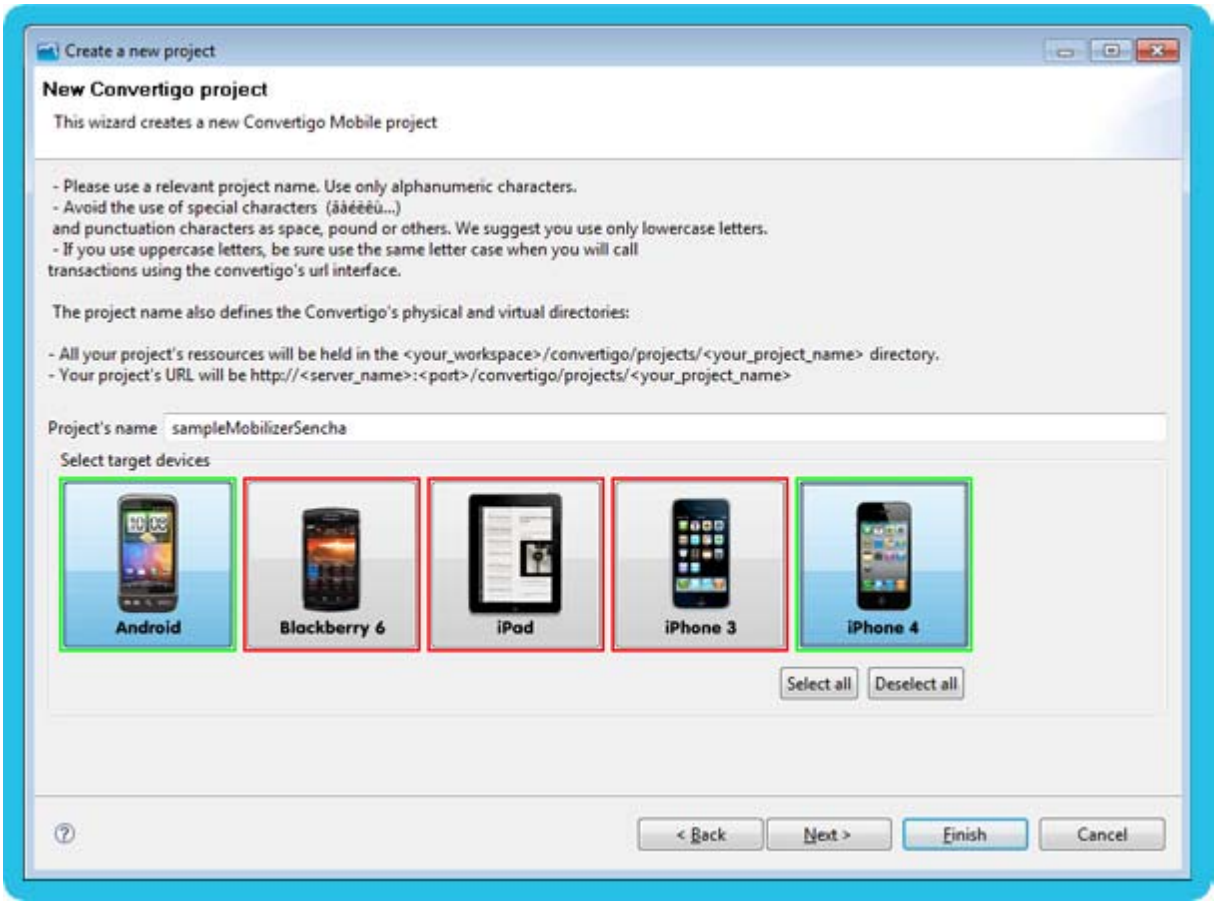


Figure 2 - 17: New project wizard - Selecting target devices

6 Click on **Next**.

7 Select the default features to be added automatically to the mobile application. In this project, we will need four features that are: a *Top toobar*, a *Search form*, a *List display* and a *Details sheet*.

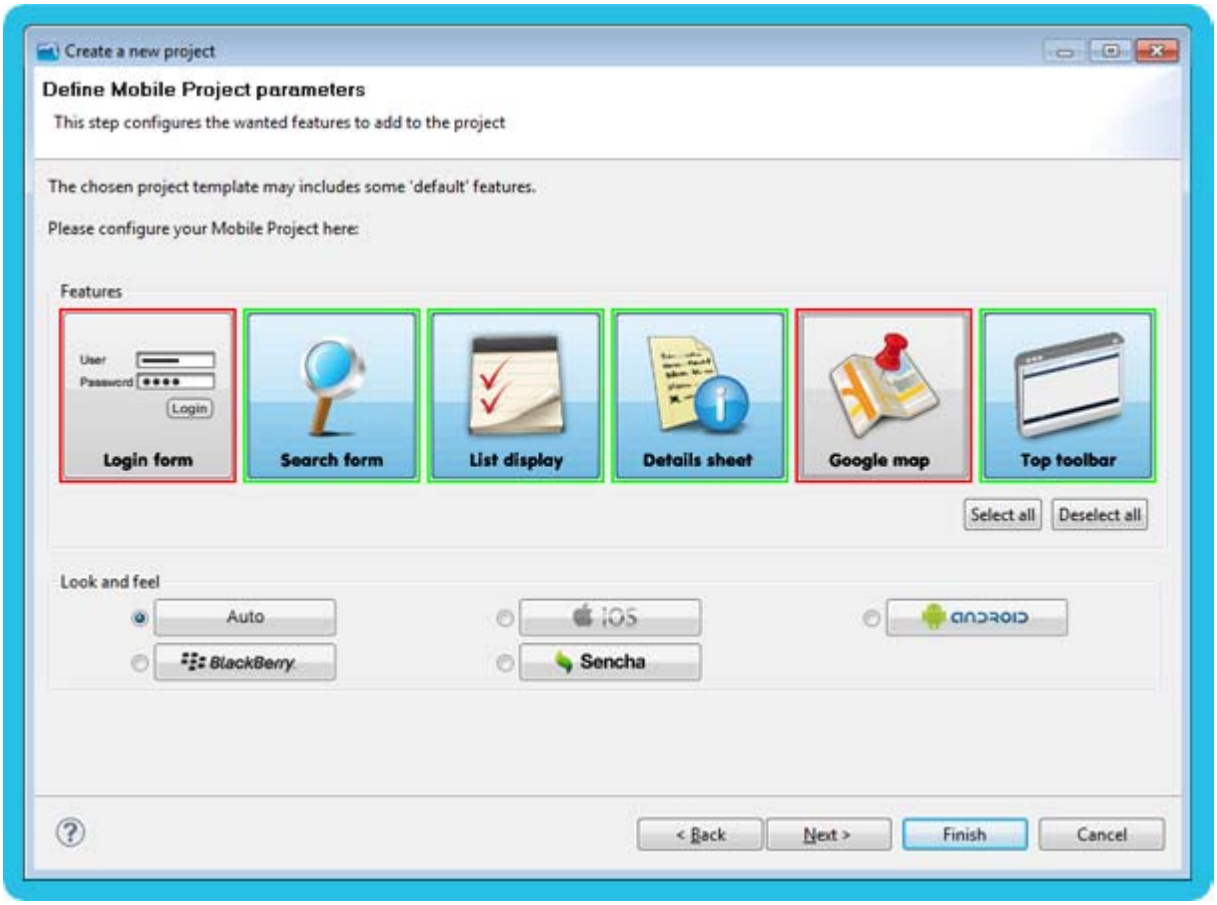


Figure 2 - 18: New project wizard - Configuring mobile features and look and feel

8 Then, select the look and feel you want for the mobile application. For example, select `Auto` look and feel, for the style to be automatically adapted on each type of device.

9 Click on **Next**.

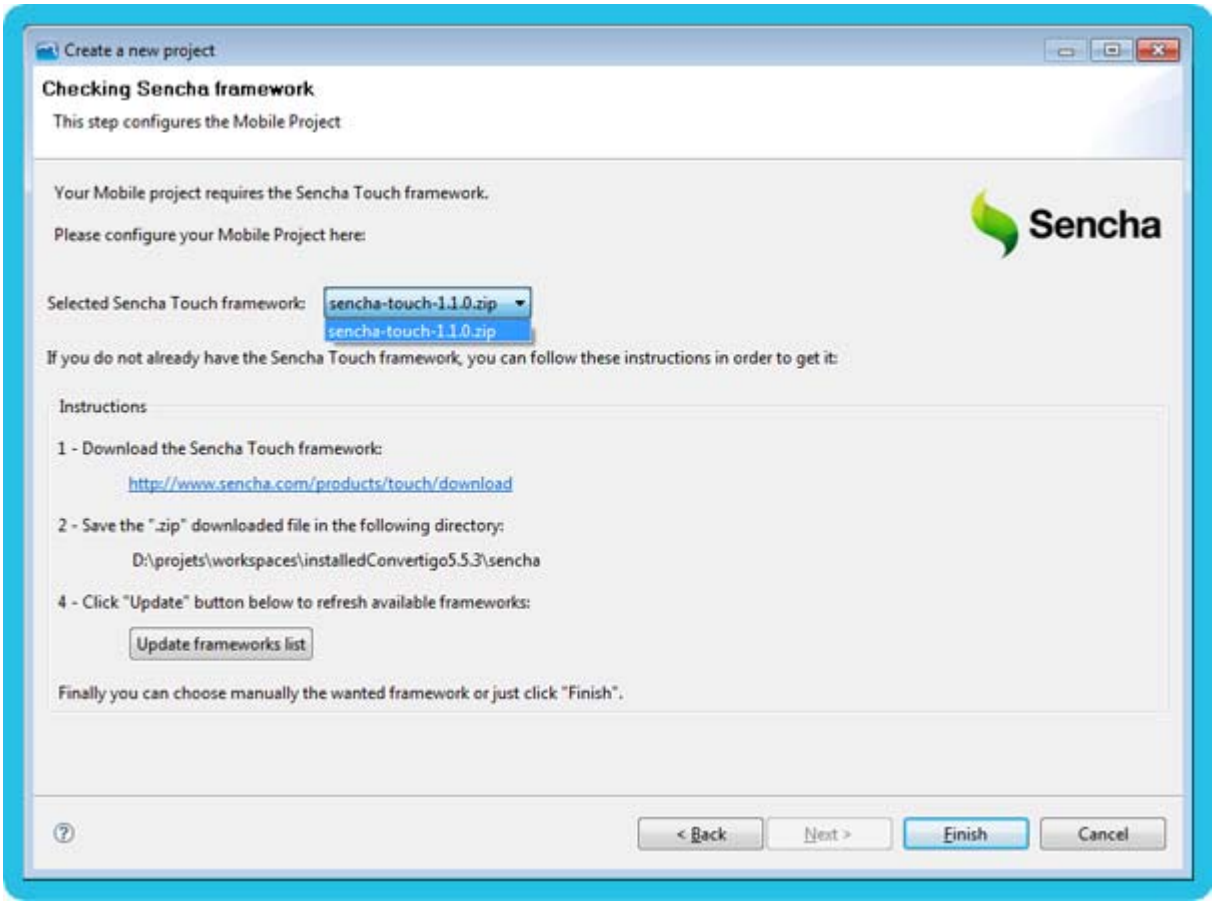


Figure 2 - 19: New project wizard - Selecting Sencha Touch framework

10 In this step, if no ***Sencha Touch*** framework is found in your workspace, you need to download it. To do so, follow the steps 6 to 8 of the procedure *"To open a Mobilizer sample project from the Studio"* on page 2 - 5, before continuing with step 11 of the procedure.

If a ***Sencha Touch*** framework is found in your workspace, select it and continue with step 11 of the procedure.

11 Click on **Finish**.

The project is automatically created in the Studio and a confirmation message is displayed, with a link to reach the test platform of the project.

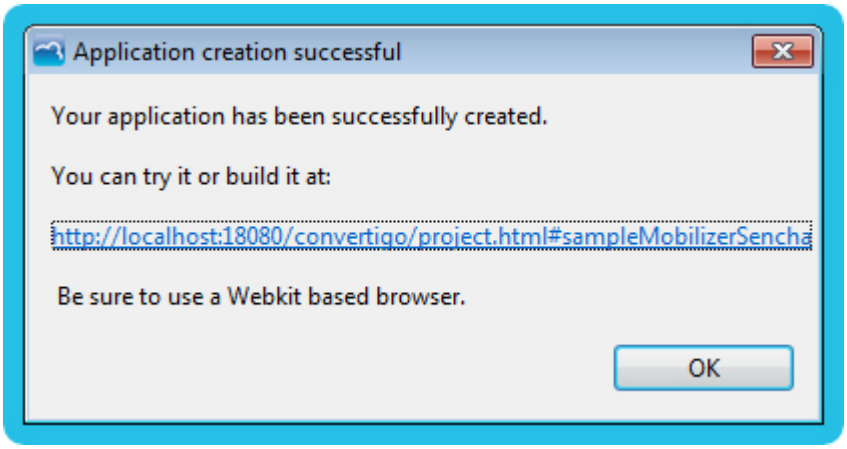


Figure 2 - 20: Project creation confirmation message

12 Click on **OK** to close the message.

The project is created in the Studio, together with a default connector `NewConnector` automatically including a default transaction `default_transaction`, two sequences `LoadList` and `Login`, and the selected mobile devices `Android` and `iPhone4`.

These objects appear in their respective folders (*Connectors, Transactions*, *Sequences* and *Mobile devices* folders) in the **Projects** view:

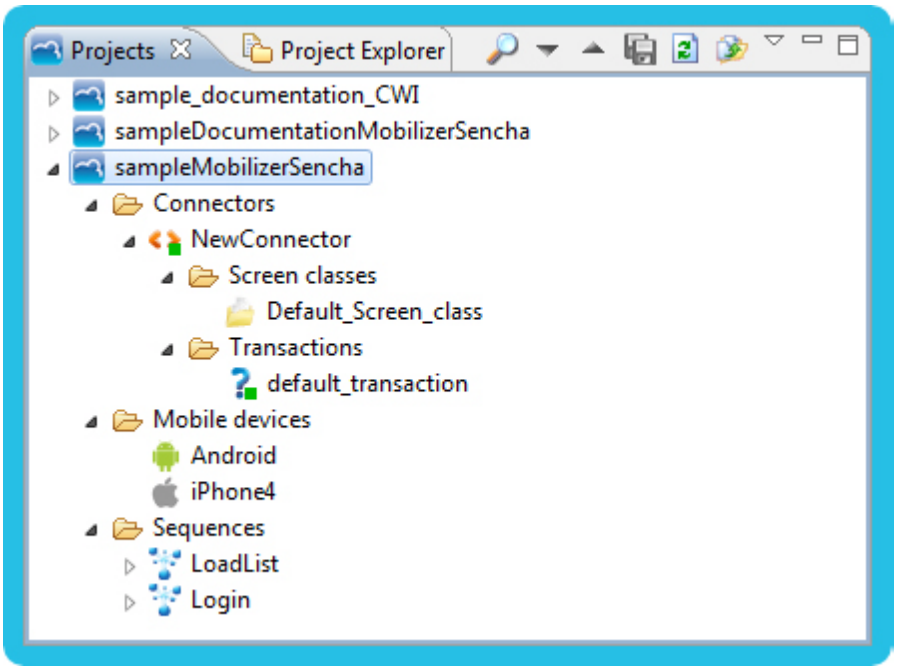


Figure 2 - 21: sampleMobilizerSencha project appearing in the Projects view

This is the starting point of the project.

The following step will consist in cleaning the generated project.

CLEANING THE BASIC PROJECT

The default Mobilizer project is providing a fully fonctional mobile application, with data provided by Convertigo to the mobile application developed using ***Sencha Touch*** framework.

For more information about the example mobile application provided by CMob template project, see the "Quick start with Convertigo Mobilizer" video on our Developer Network: http://www.convertigo.com/en/products/videos/288-quick-start-with-convertigo-mobilizer.html

The two sequences automatically created in the project are used to generate data to feed the basic template features. In your project, these sequences won't be needed as the data will be retrieved from the `sample_documentation_CWI` project's transactions.

To remove unused sequences

1 In the **Projects** view, right-click on the `LoadList` sequence and select **Delete**.

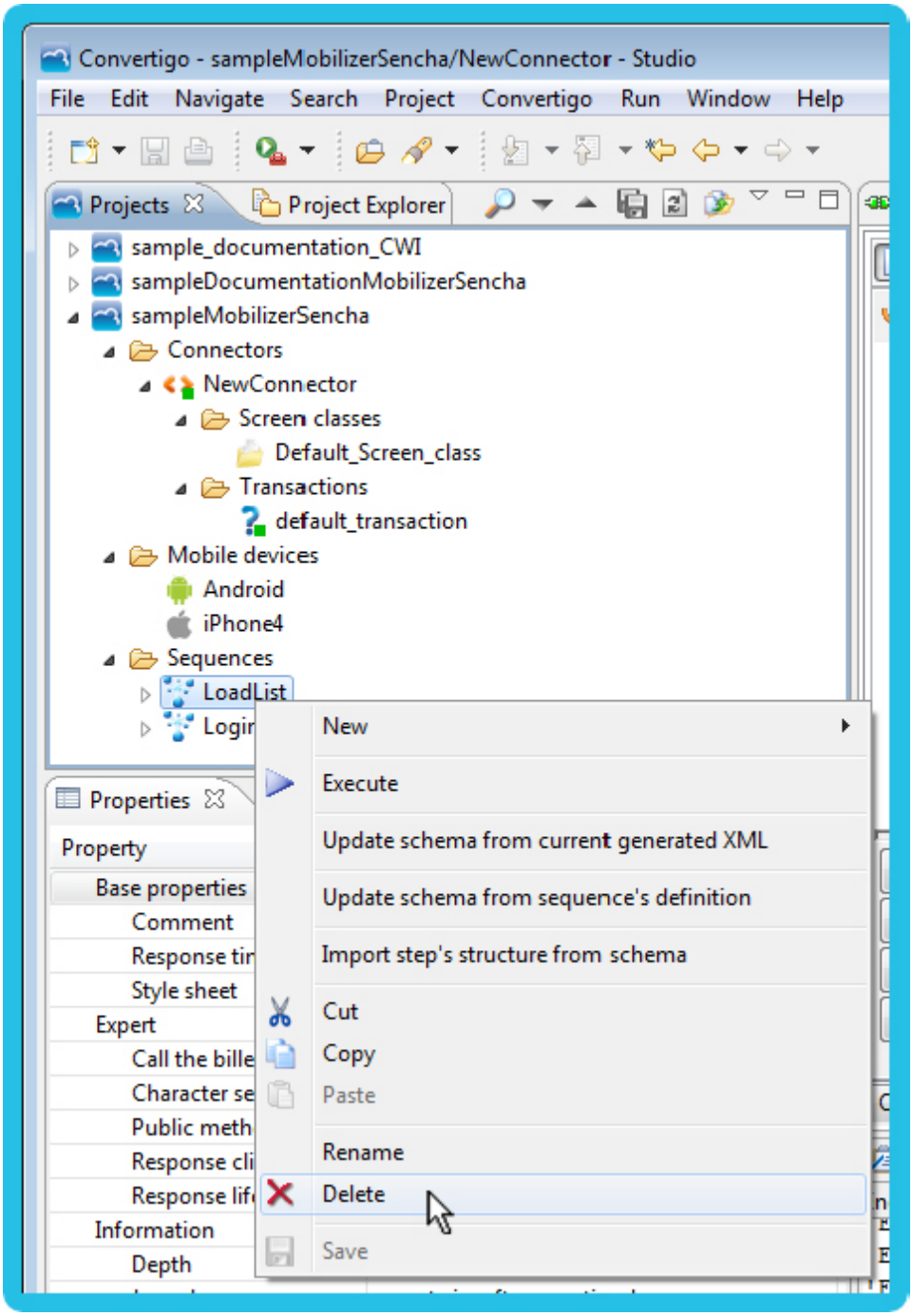


Figure 2 - 22: Deleting unused sequence

A deletion confirmation message appears:

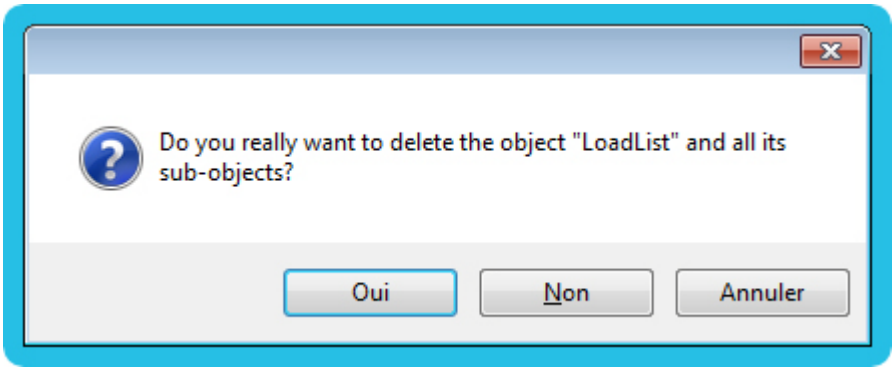


Figure 2 - 23: Deletion confirmation message

2 Click on **Yes**.

3 Repeat previous steps for the second sequence `Login`.

The Mobilizer project is updated in the **Projects** view:

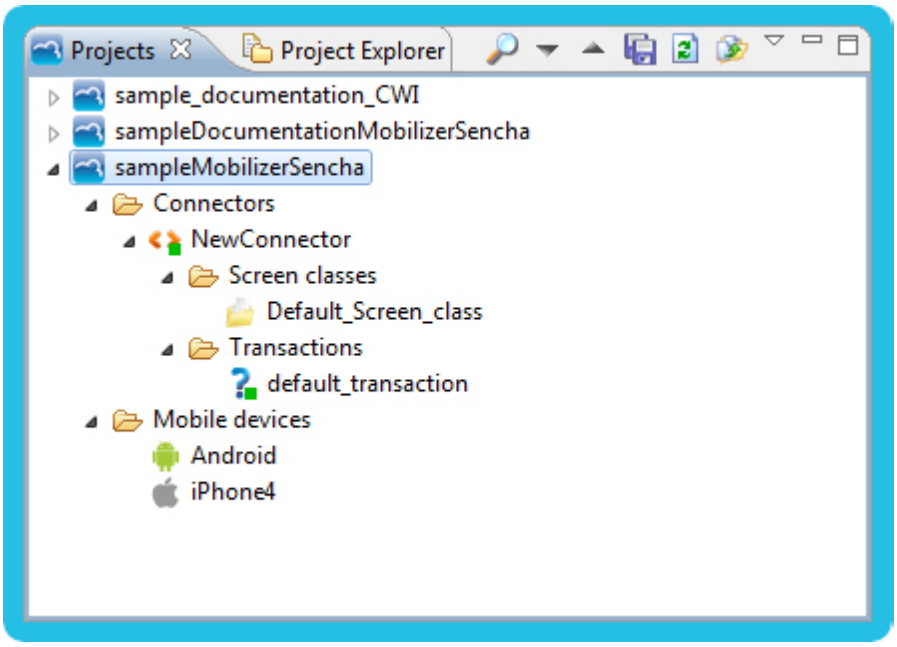


Figure 2 - 24: Cleaned Mobilizer project

4 Save your project by clicking on or by pressing `Ctrl + S`.

The basic template features use resources available in the project: the `LoadList` sequence result is based on an hard-coded XML file containing data. In your project, as the sequence has been removed, the XML file won't be needed anymore.

To remove unused resources

1 In the **Projects** view, click on the **Project Explorer** tab.

2 Expand the project folder:

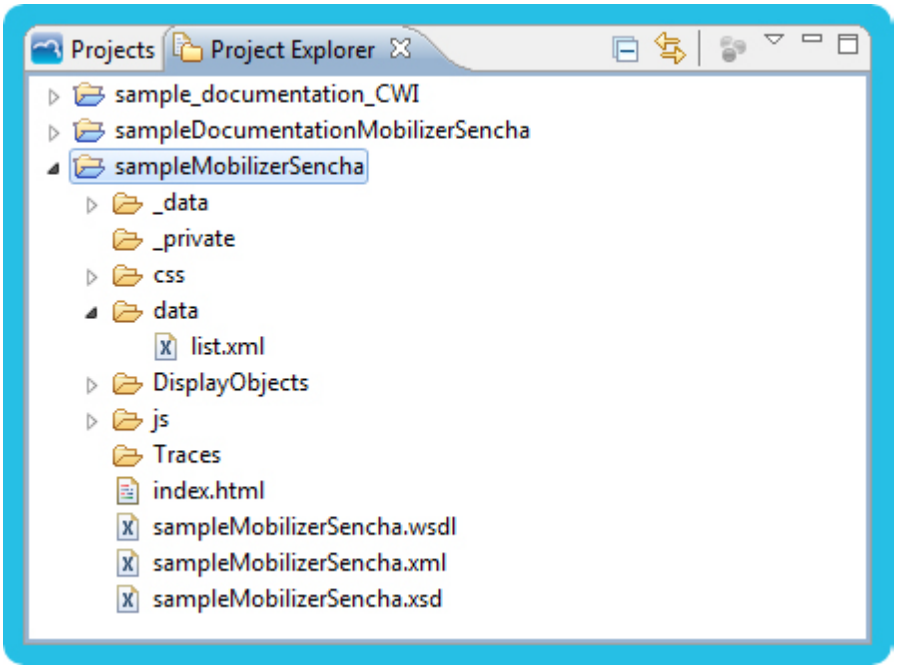


Figure 2 - 25: Project folder in Project Explorer

In the `data` folder, you can see the `list.xml` file used by the previously removed sequence.

3 Right-click on the `data` folder and select **Delete**.

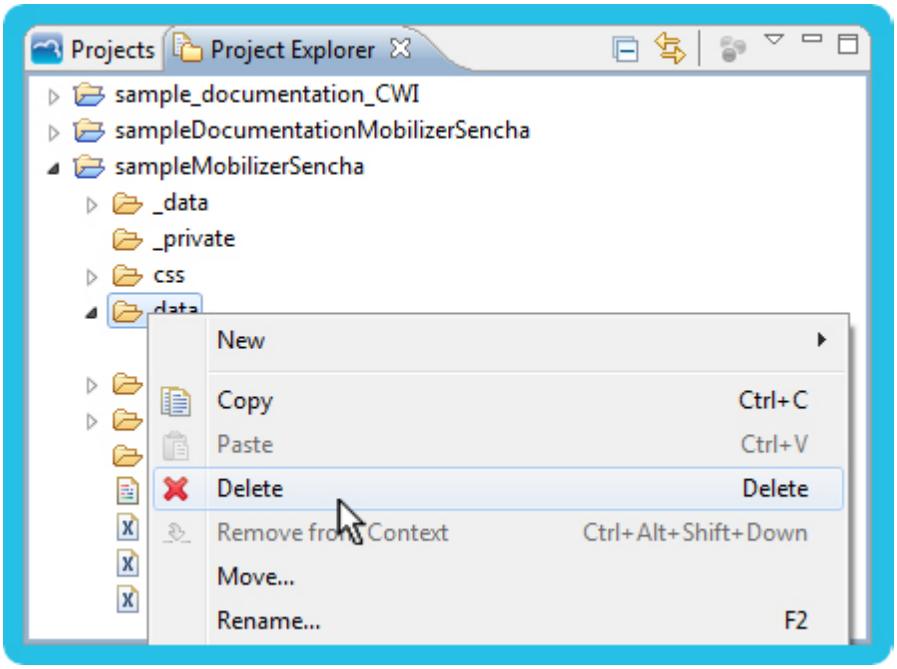


Figure 2 - 26: Deleting unused resources

Beware not to select `_data` folder instead of `data`, this private directory is used by Convertigo to save elements of your project.

A deletion confirmation message appears:

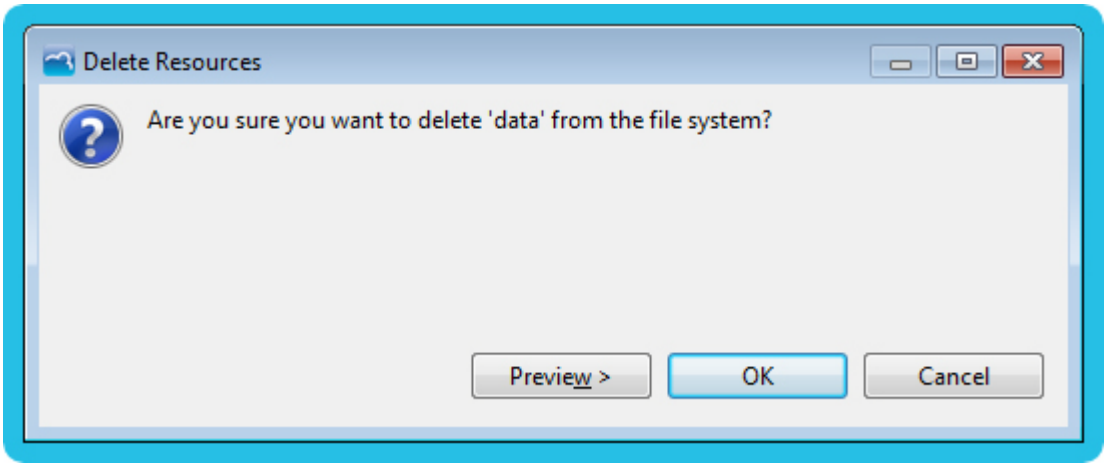


Figure 2 - 27: Deletion confirmation message

4 Click on **OK**.

The project resources are updated in the **Project Explorer** view:

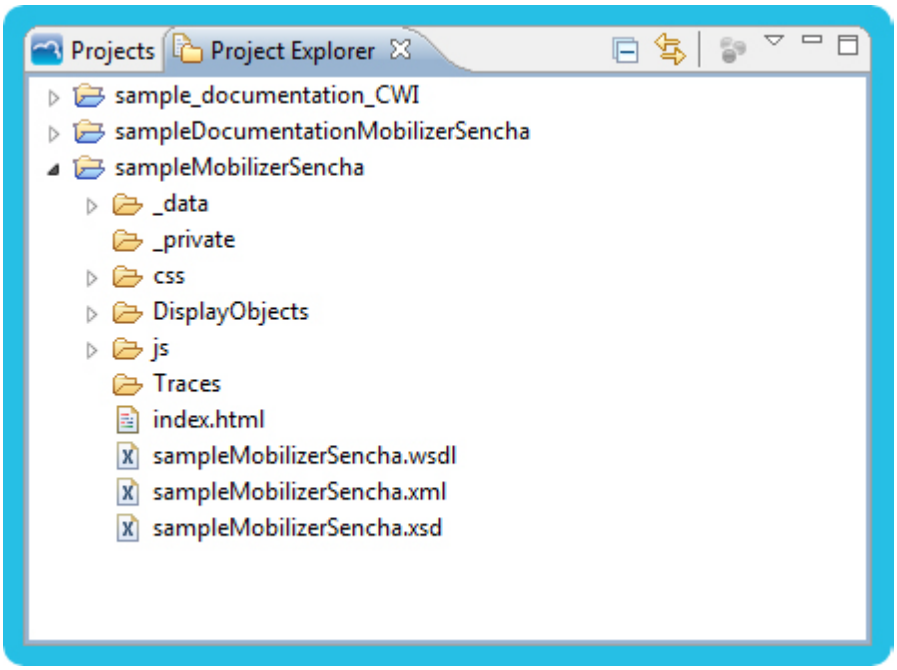


Figure 2 - 28: Cleaned project resources

5 Click on the **Projects** tab to switch back to the **Projects** view.

To clean the Studio interface

As previously said when opening the sample project, the connector of the Mobilizer project is not used in this mobile application. You can close its editor by clicking on button.

WHAT COMES NEXT?

After creating and configuring the project, the next step is the creation of the mobile user interface using the ***Sencha Touch*** framework. This framework is based on JavaScript, HTML5 and CSS3 languages.

2.3.2 Creating the mobile user interface

This section presents the creation of the mobile application interface. It starts with a quick presentation of the project environment where we will create in a second phase the mobile user interface.

- Mobile environment presentation
- Creating the mobile user interface

MOBILE ENVIRONMENT PRESENTATION

First step is to present the project environment using ***Sencha Touch*** framework, in which JavaScript, HTML and CSS files are written to create the mobile application.

1 In the **Projects** view, click on the **Project Explorer** tab.

2 Expand the project folder:

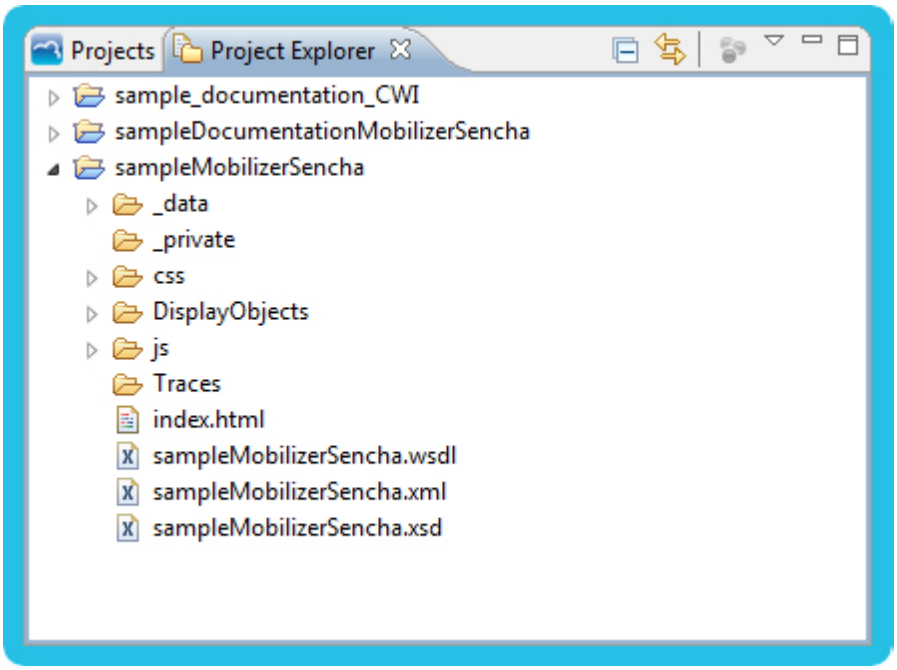


Figure 2 - 29: Project folder in Project Explorer

3 Expand `DisplayObjects` folder, then `mobile`:

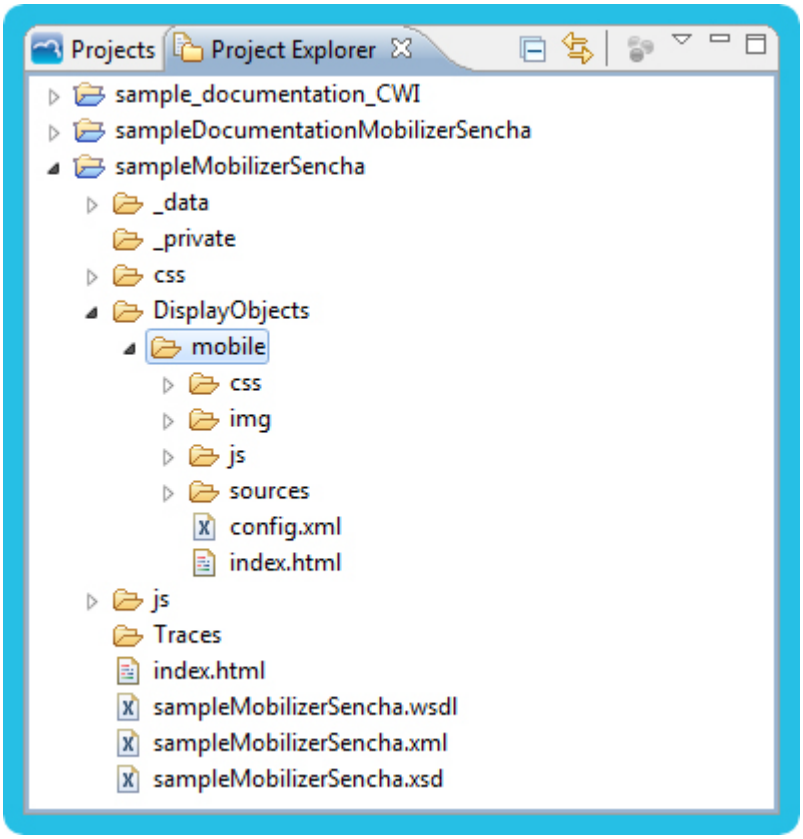


Figure 2 - 30: DisplayObjects folder and mobile folder and subfolders

The `DisplayObjects` folder contains a folder named `mobile`. It is the workspace of development, containing all sources of the mobile application interface.

The `mobile` folder contains two files named `index.html` and `config.xml` and four directories, which are `css`, `img`, `js` and `sources`. Let's present these subfolders and files.

CSS DIRECTORY

This folder contains the style sheets (CSS files) corresponding to selected look and feels in step 8 of project's creation procedure:

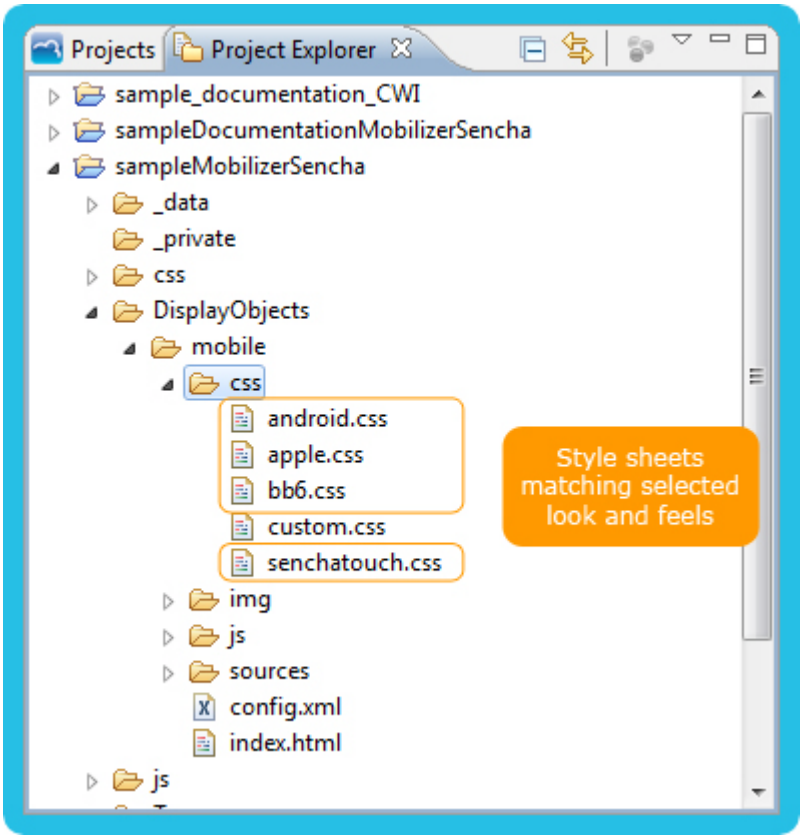


Figure 2 - 31: Project look and feels style sheets

In this case, we selected **Auto** look and feel so all look and feels matching style sheets are present in the `css` directory. At runtime, only one of the CSS files is active, depending on which mobile device the application runs.

*If we selected the **Sencha** look and feel for example in step 8 of project's creation, we would have only Sencha look and feel matching style sheet (`senchatouch.css`) in the `css` directory.*

A `custom.css` style sheet is also present in the `css` folder. It contains sample custom CSS styles developed in the context of the functional sample included in the default Mobilizer project. This file can be edited to customize CSS styles necessary for the mobile application.

IMG DIRECTORY

This directory contains default icons and phone startup images. It would contain all mobile application images used in the mobile interface.

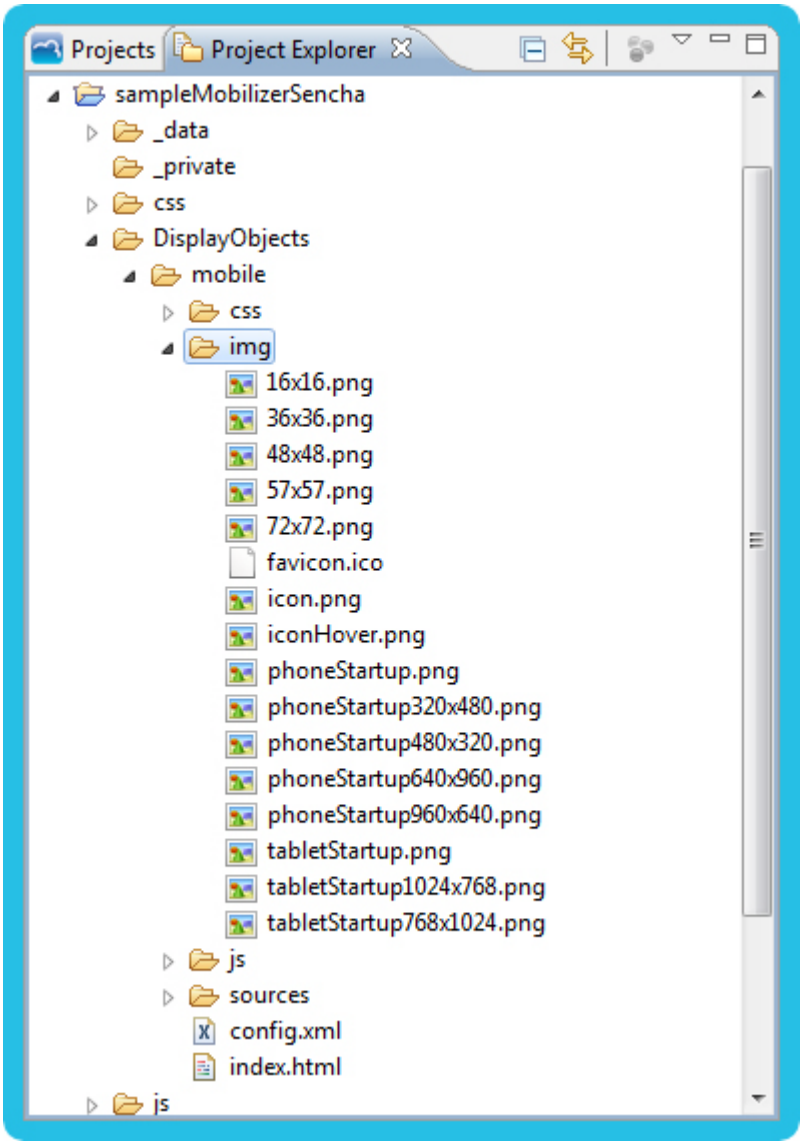


Figure 2 - 32: Images in `img` directory

Those image files are referenced in other files, their purpose will be explained at this time.

JS DIRECTORY

This directory contains JavaScript library files:

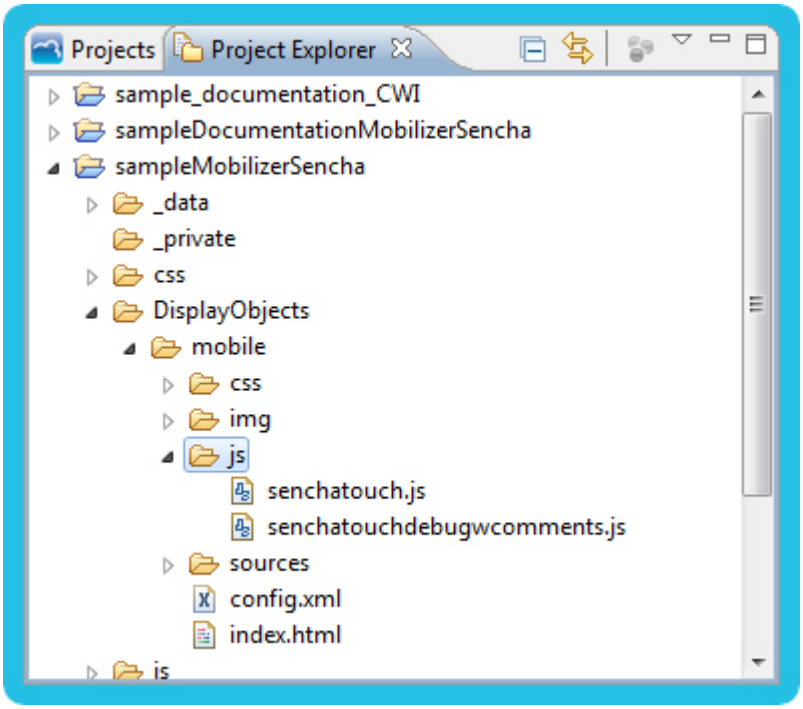


Figure 2 - 33: JavaScript library files

Two files are present in this directory:

- the ***Sencha Touch*** library: `senchatouch.js` file;
- the ***Sencha Touch*** debug library: `senchatouchdebugwcomment.js` file.

They are both retrieved from the ***Sencha Touch*** framework ZIP file previously downloaded and set in Convertigo Studio workspace directory.

By default, the project uses the debug library in order to be able to step in Sencha commented source code while developing the mobile application.

SOURCES DIRECTORY

This directory contains the default source files of template application pages, corresponding to features chosen in step 7 of the project's creation procedure:

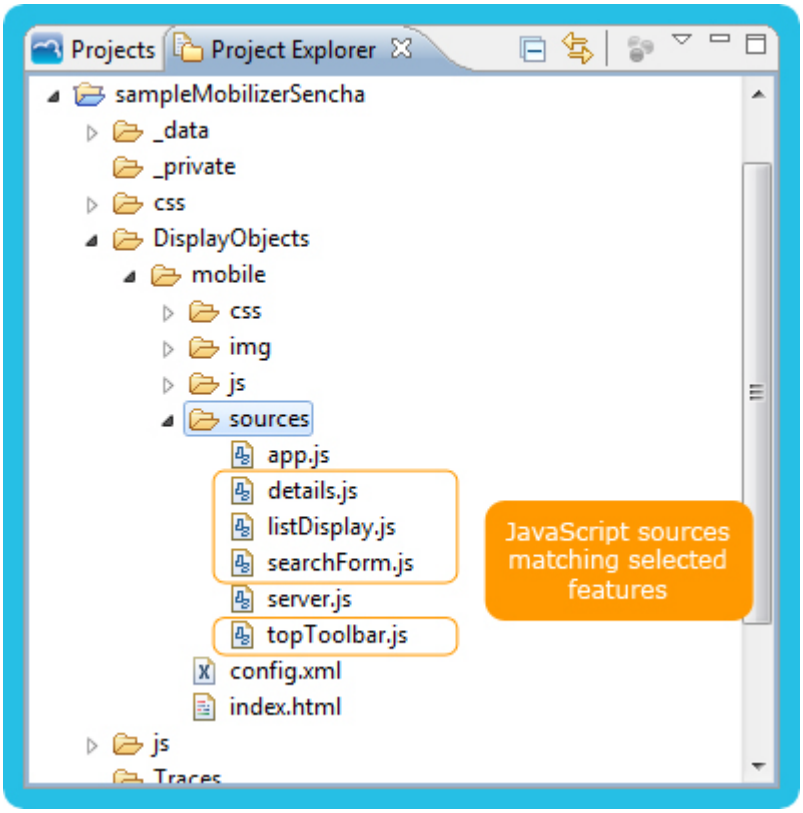


Figure 2 - 34: Sources of the mobile application pages

In this case, we selected four features: **Search form**, **List display**, **Details sheet** and **Top toolbar**. We then have four corresponding files: `searchForm.js`, `listDisplay.js`, `details.js` and `topToolbar.js`.

Other JavaScript files are also present in `sources` folder:

- `app.js`: which is the application main JavaScript file,
- `server.js`: used to define the server endpoint used for ***PhoneGap build***.

We recommend not to update this file manually, it will be automatically modified during build phase.

Let's have a look to the `app.js` file.

1 Double-click on the `app.js` file to open it in the editor:

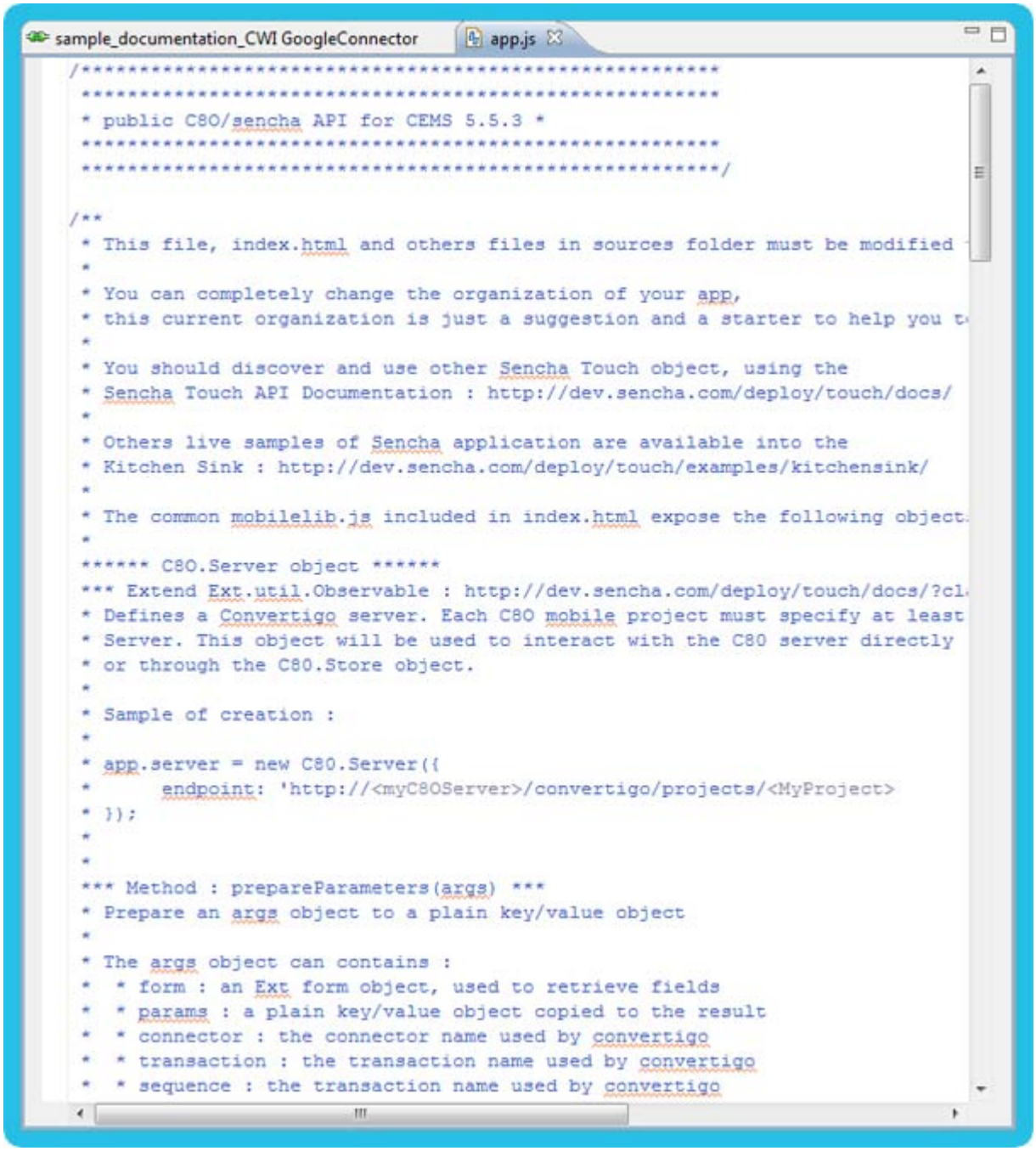


Figure 2 - 35: Main JavaScript file app.js in editor

This main JavaScript file `app.js` contains the documentation of the **Convertigo mobilelib** API in comments. It is the first thing you see when opening the file in the editor.

2 Scroll down in source code. An `app` namespace is declared for the mobile application. It declares a global variable, named app, that will be used to store all the objects of the application:

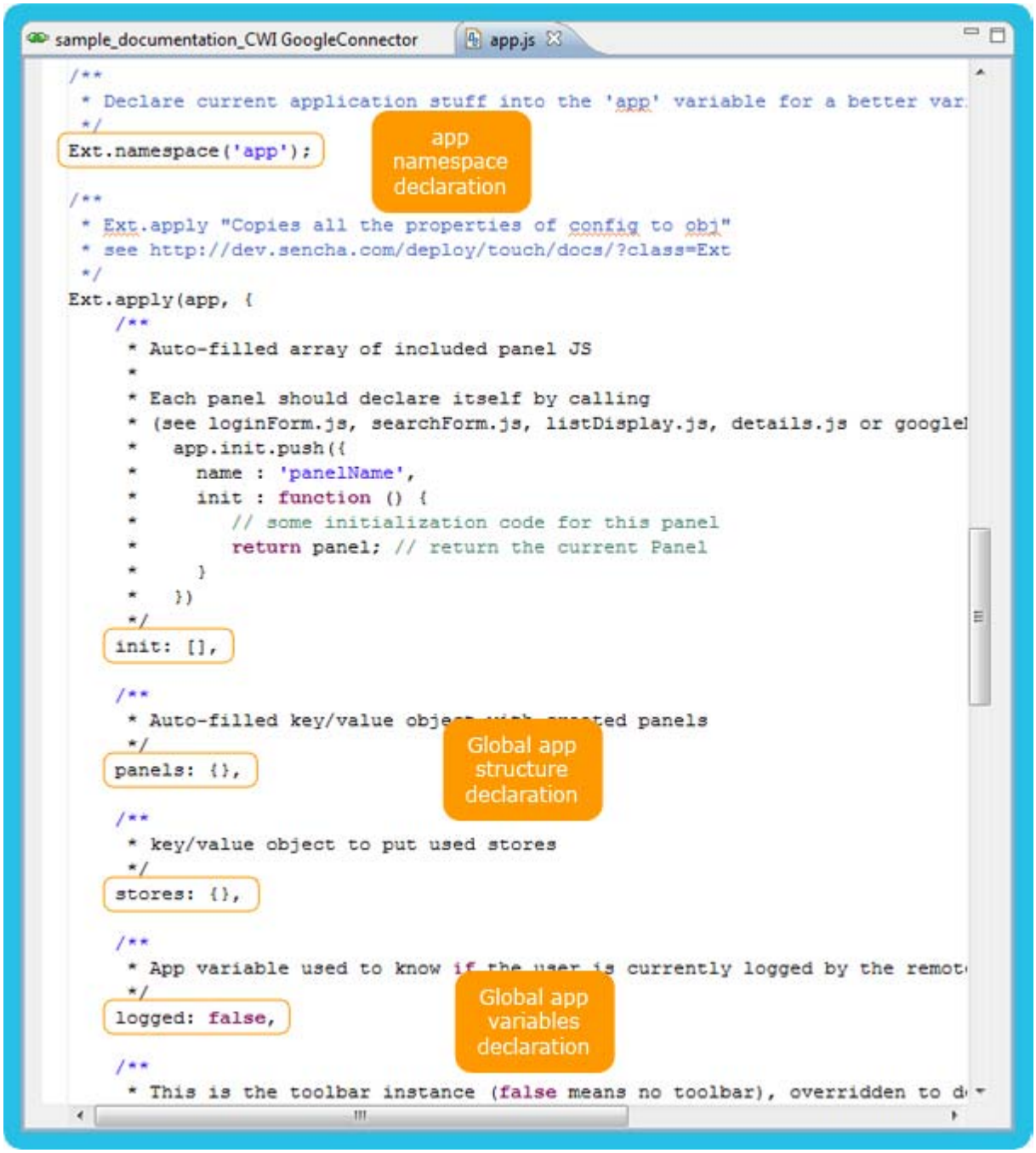
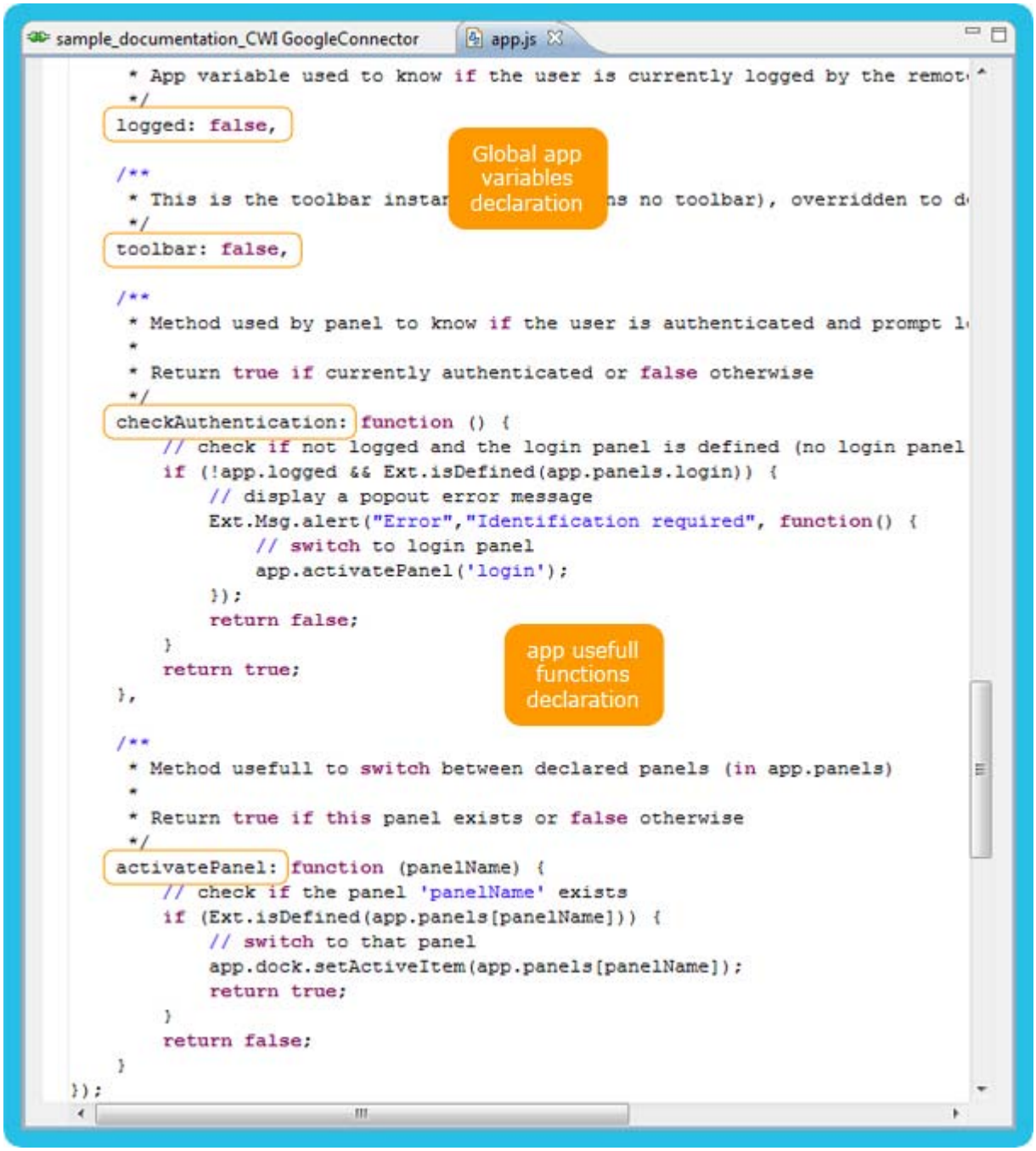


Figure 2 - 36: Main app.js: app global structures and variables

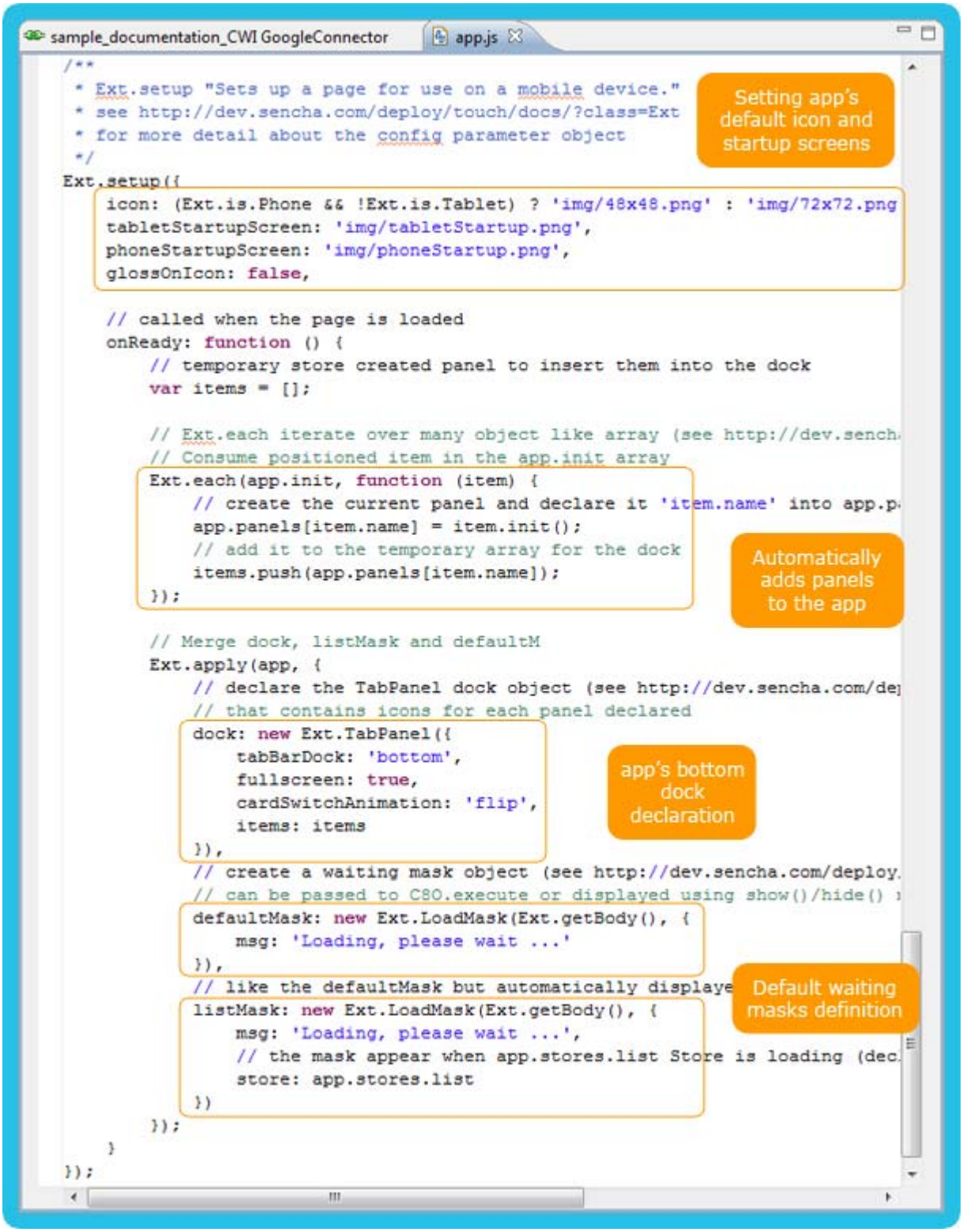
3 Then, the `Ext.apply` method is used to declare the structure of the `app` object. The `app` object is a defined as a structure containing several key/value couples used to store all needed objects of the application as well as variables and usefull functions. Here, keys are defined with possibly a default value for the variable and function definitions:




```
    * App variable used to know if the user is currently logged by the remot
    */
   logged: false,

   /**
    * This is the toolbar instar                ns no toolbar), overridden to d
    */
   toolbar: false,

   /**
    * Method used by panel to know if the user is authenticated and prompt l
    *
    * Return true if currently authenticated or false otherwise
    */
   checkAuthentication: function () {
       // check if not logged and the login panel is defined (no login panel
       if (!app.logged && Ext.isDefined(app.panels.login)) {
           // display a popout error message
           Ext.Msg.alert("Error","Identification required", function() {
               // switch to login panel
               app.activatePanel('login');
           });
           return false;
       }
       return true;
   },

   /**
    * Method usefull to switch between declared panels (in app.panels)
    *
    * Return true if this panel exists or false otherwise
    */
   activatePanel: function (panelName) {
       // check if the panel 'panelName' exists
       if (Ext.isDefined(app.panels[panelName])) {
           // switch to that panel
           app.dock.setActiveItem(app.panels[panelName]);
           return true;
       }
       return false;
   }
});
```

Figure 2 - 37: Main app.js: app global variables and usefull functions

4 Scroll down again to access default source code corresponding to the basic app set up for template project. It calls the `Ext.setup` function that sets up a page to be used on a touch-enabled device:




```
/**
 * Ext.setup "Sets up a page for use on a mobile device."
 * see http://dev.sencha.com/deploy/touch/docs/?class=Ext
 * for more detail about the config parameter object
 */
Ext.setup({
    icon: (Ext.is.Phone && !Ext.is.Tablet) ? 'img/48x48.png' : 'img/72x72.png
    tabletStartupScreen: 'img/tabletStartup.png',
    phoneStartupScreen: 'img/phoneStartup.png',
    glossOnIcon: false,

    // called when the page is loaded
    onReady: function () {
        // temporary store created panel to insert them into the dock
        var items = [];

        // Ext.each iterate over many object like array (see http://dev.sench
        // Consume positioned item in the app.init array
        Ext.each(app.init, function (item) {
            // create the current panel and declare it 'item.name' into app.p
            app.panels[item.name] = item.init();
            // add it to the temporary array for the dock
            items.push(app.panels[item.name]);
        });

        // Merge dock, listMask and defaultM
        Ext.apply(app, {
            // declare the TabPanel dock object (see http://dev.sencha.com/de
            // that contains icons for each panel declared
            dock: new Ext.TabPanel({
                tabBarDock: 'bottom',
                fullscreen: true,
                cardSwitchAnimation: 'flip',
                items: items
            }),
            // create a waiting mask object (see http://dev.sencha.com/deploy.
            // can be passed to C8O.execute or displayed using show()/hide()
            defaultMask: new Ext.LoadMask(Ext.getBody(), {
                msg: 'Loading, please wait ...'
            }),
            // like the defaultMask but automatically displaye
            listMask: new Ext.LoadMask(Ext.getBody(), {
                msg: 'Loading, please wait ...',
                // the mask appear when app.stores.list Store is loading (dec
                store: app.stores.list
            })
        });
    }
});
```

Figure 2 - 38: Main app.js: app startup properties definitions

This function allows to set various startup properties and behaviors of the mobile application, in our case:

- **icon** property: specifies the image used as application's default icon when setting a desktop shortcut (for *iOS* devices);
- **tabletStartupScreen** property and **phoneStartupScreen** property: specify the

images used as application's startup screen on tablet / phone devices;

- **glossOnIcon** property: specifies whether you want a gloss effect to be applied to the default icon. In this case, the value is set to false indicating not to add gloss effect to the default icon.
- **onReady** property: specifies the function to run when the browser's Document Object Model (DOM) is ready after the application HTML file has loaded. This function already includes to the app:
  - the *panels* corresponding to previously selected features,
  - the *bottom dock* including button icons to navigate through panels,
  - default *waiting masks* that could be activated when loading data from a Convertigo transaction's or sequence's execution.

*For detailed information about **Sencha Touch** API, including all the properties that could be adedd to these pieces of source code, refer to the Sencha Touch API documentation at: http://dev.sencha.com/deploy/touch/docs/*

Other JavaScript files from this folder will be described when editing the code of the mobile application user interface.

INDEX.HTML FILE

The role of this HTML index is to link all the necessary files to the mobile application:

- the application `favicon`, defining the navigation bar icon for using the web version of the mobile application, and used as application's default icon when setting a desktop shortcut (for *Android* devices),
- the application CSS files,
- the ***Sencha Touch*** library JS files,
- the **Convertigo mobilelib** library JS file,
- and the application JavaScript files declaring features panels.

Figure 2 - 39: Links to application resources in index.html

CONFIG.XML FILE

This is the configuration file for ***PhoneGap build*** process. It contains, in a first part a general description of the mobile application to build:

- the name, description and author of the mobile application,
- the list of plateforms on which the mobile application should be built (by default, all ***PhoneGap build*** available plateforms are listed, even though you may have chosen a shorter list of mobile devices in the Convertigo project),
- the list of all available icon images with their size. ***PhoneGap build*** allows to dynamically use the correct size matching image as icon for the application on each type of device, depending on the size of icon on each device,
- the list of all available splashscreen images with their size. ***PhoneGap build*** allows to dynamically use the correct size matching image as splashscreen for the application startup on each type of device, depending on the resolution of each device.

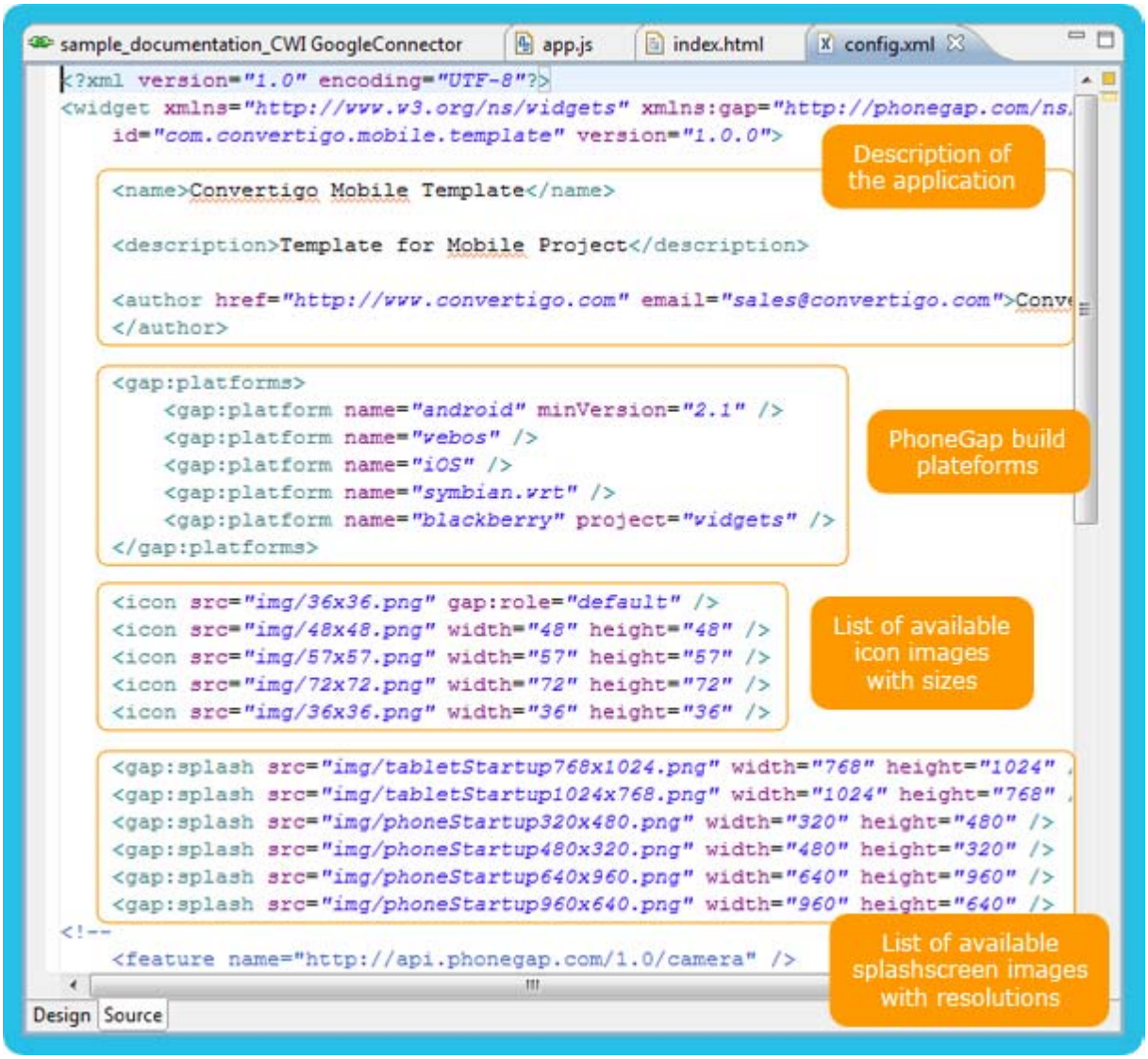


Figure 2 - 40: Mobile application general description in config.xml

In a second part, it contains the description of accesses and authorizations required for the mobile application to run on mobile devices:

- the list of phone features which access is required by the application to run (by default a lot of features are not enabled as required, as the application doesn't need them),
- the list of BlackBerry-specific features which access is required by the application to run correctly on BlackBerry devices,
- the list of URI domains the application can access, by default, it is set to access both external (HTTP protocol) and local (file protocol) URIs, and for local, both on the device storage and on the SDCard memory,
- the root page of the application, always set to `index.html` file, which is the page including all resources and libraries and thus starting the application.

Figure 2 - 41: Mobile application required accesses and root in config.xml

Now that you are introduced to the **Convertigo Mobilizer** development environment, you can close all opened files editors. Next step is to create the user interface of your mobile application.

CREATING THE MOBILE USER INTERFACE

This section describes step by step how to implement the user interface of your mobile application using ***Sencha Touch*** framework in Mobilizer environment, previously described in *"Mobile environment presentation"* on page 2 - 24.

First step is to create the first mobile page of your mobile application: the search form. It should contain:

- an input text field to type in the searched keyword,
- a number field to select the number of result pages to accumulate,
- a validation button to launch the research.

To create a form page

In previously selected features, the *Search form* feature was chosen. A JavaScript file was created in the project environment to implement this page: `searchForm.js`.

1 In the **Project Explorer** view, expand the `sources` folder and double-click on the `searchForm.js` file to open it in the editor.

Default source code found in this file corresponds to the basic Mobilizer template application. It is an example of a room booking form panel containing:

- a title,
- the inclusion of the top toolbar,
- a dock icon,
- a fieldset with four fields:
  - a combo box for selecting country,
  - a text field for setting a postal code,
  - a date picker to select a date,
  - a combo box to select the type of room,
- and a container with a button to launch the booking research.

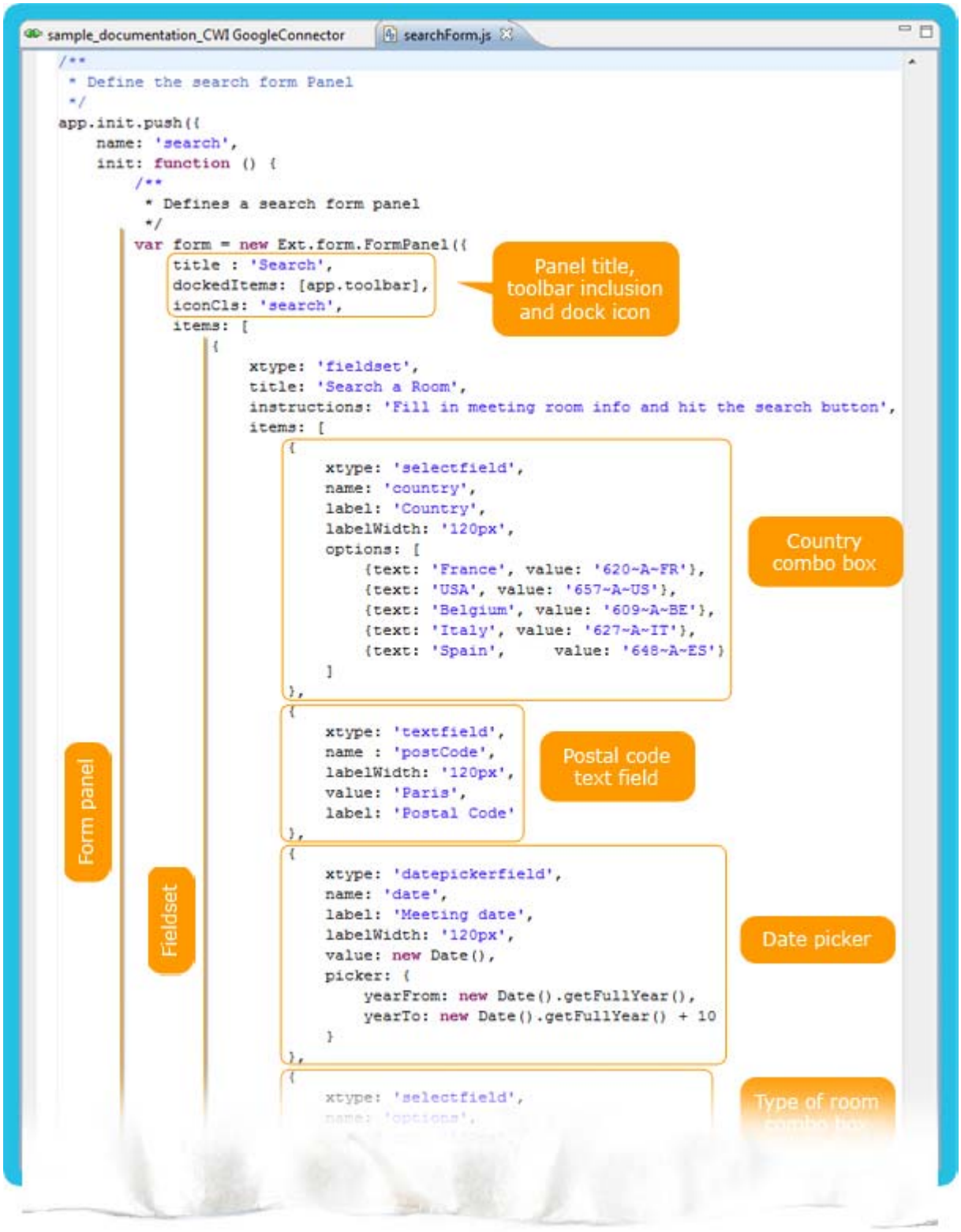


Figure 2 - 42: Default searchForm.js file displayed in editor - part 1

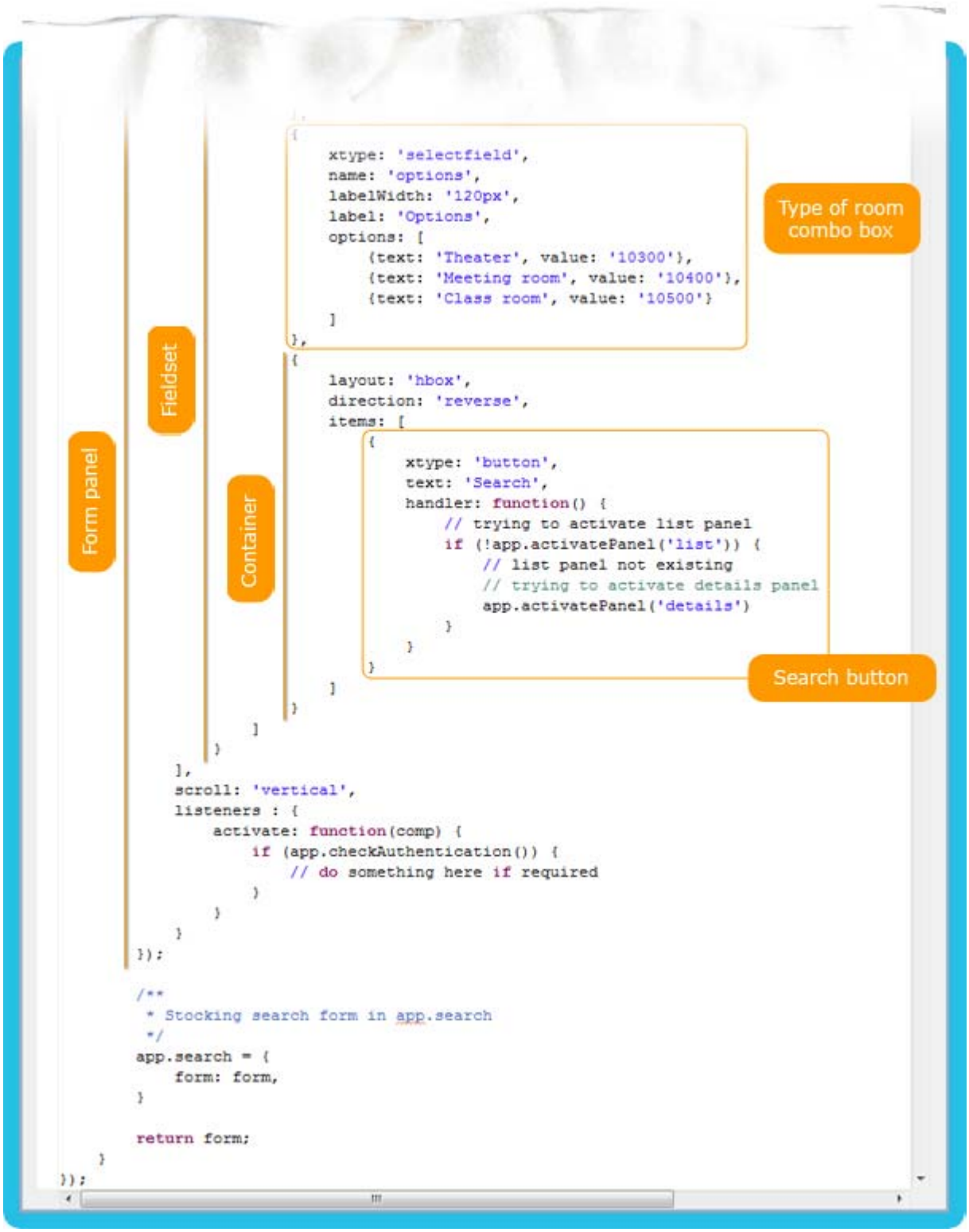


Figure 2 - 43: Default searchForm.js file displayed in editor - part 2

The default source code has to be modified according to the needs of the currently developed application.

2 Modify the code up to get the following source code:

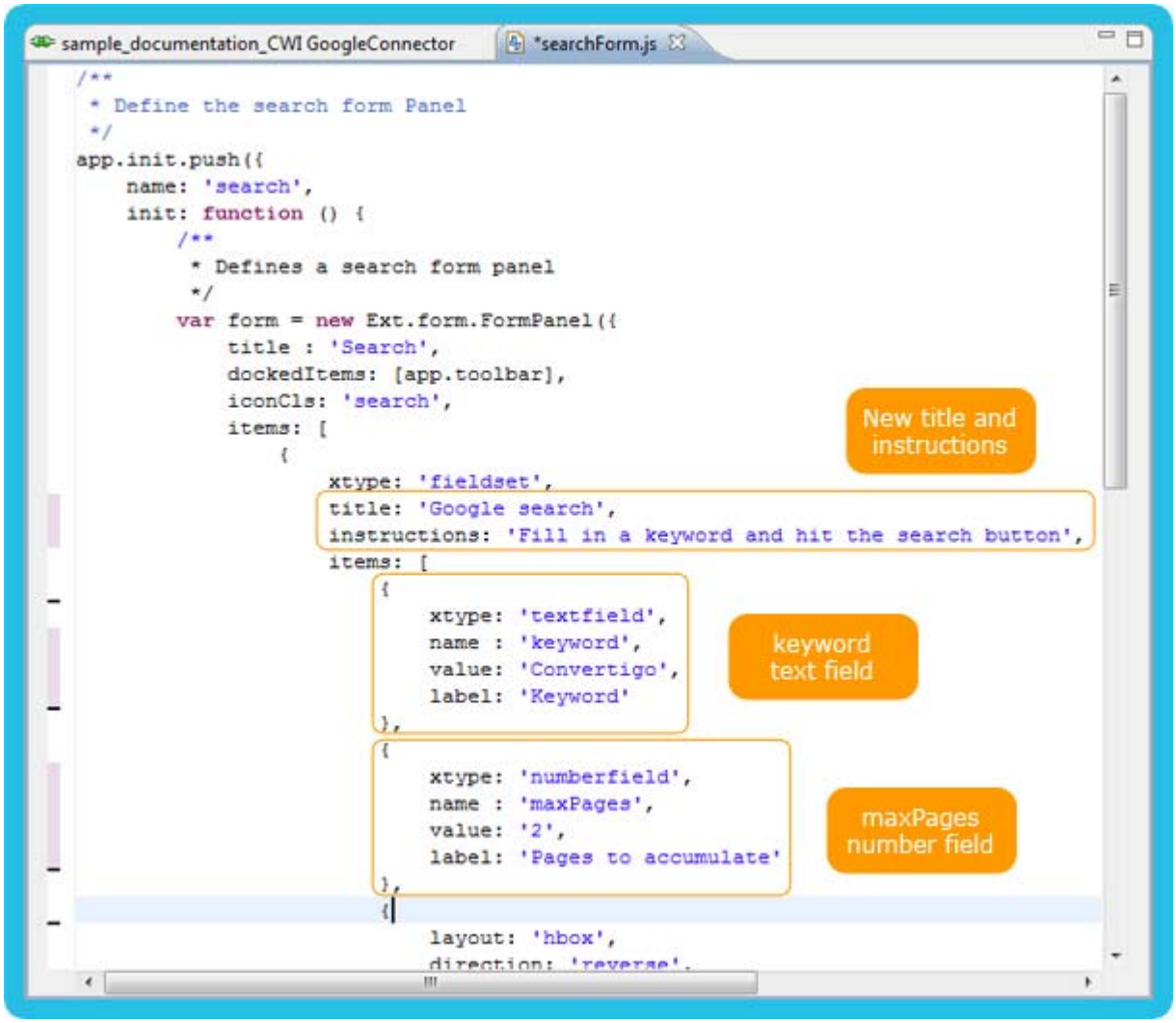


Figure 2 - 44: Updating searchForm.js file in editor

You can find the complete edited `searchForm.js` file in the example project `sampleDocumentationMobilizerSencha` resources. You can copy and paste source code from it to save time.

This updated source code allows to create:

- an **input field**, named `keyword`, with a default value that is "`convertigo`";
- a **number field**, named `maxPages`, with a default value that is "2";
- the already existing **search button** with a non-modified handler function for the moment. This function will implement the actions to be executed when a user clicks on the search button.

Beware! Follow the steps of the tutorial, do not copy and paste the complete source code. At this stage, the `handler` function of the `search button` is not yet updated.

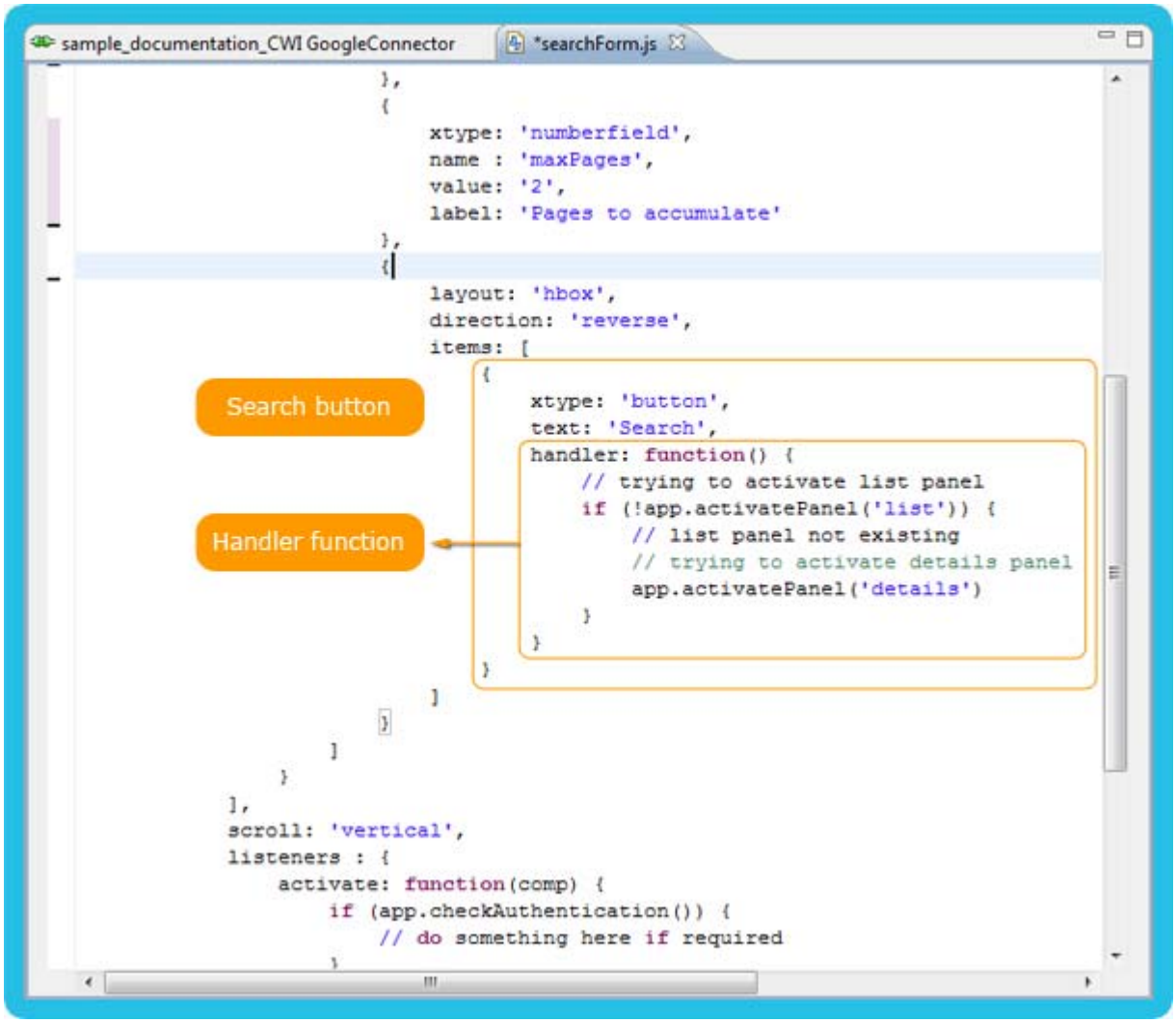


Figure 2 - 45: Non-updated search button in searchForm.js file

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

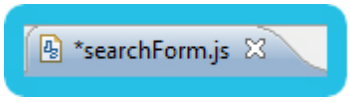


Figure 2 - 46: Edited searchForm.js file

3 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

Now, let's have a look at the **search button** handler function. Default source code implemented for the basic Mobilizer template application tries to activate the list panel on the button's click, and if not successful, activates the details panel.

This code allows the mobile application to continue functionning even in absence of the list panel, i.e. if the *List display* feature has not been selected by the user at the project creation.

In our case, we do have selected the *List display* feature, so the list panel exists in our mobile application and it is the only panel to be activated on the **search button**'s click.

4 Modify the handler function code up to get the following source code:

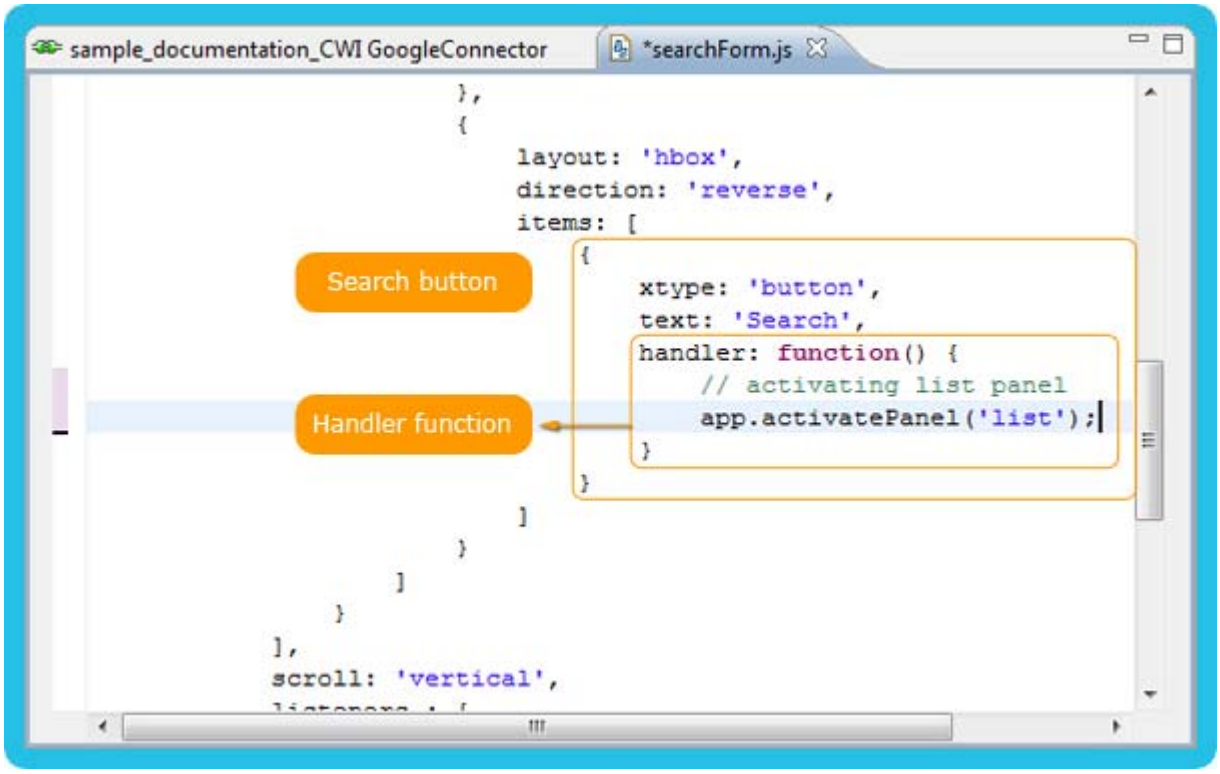


Figure 2 - 47: Updating search button handler function

You can find the complete edited `searchForm.js` file in the example project `sampleDocumentationMobilizerSencha` resources. You can copy and paste source code from it to save time.

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

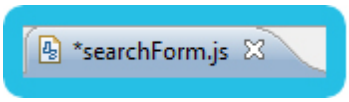


Figure 2 - 48: Edited searchForm.js file

5 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

WHAT COMES NEXT?

The default code of each feature also adds the toolbar defined when choosing the *Top toobar* feature. Next step is to edit this toolbar source code to make it meet our application's needs.

To create a top toolbar

A JavaScript file was created in the project environment to implement the *Top toolbar* feature: `topToolbar.js`.

1 In the **Project Explorer** view, expand the `sources` folder and double-click on the `topToolbar.js` file to open it in the editor. Default source code found in this example file contains only a text field including a title:

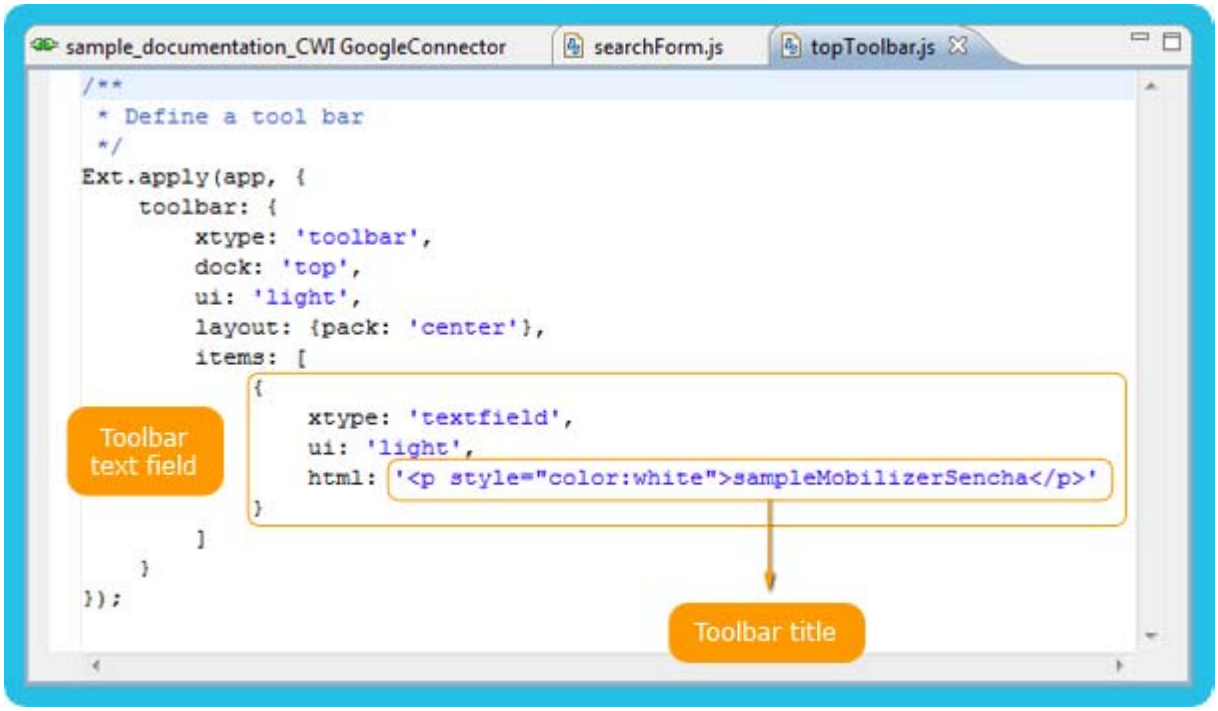


Figure 2 - 49: Default topToolbar.js file displayed in editor

The toolbar title is implemented thanks to HTML code and includes the name of the project in a paragraph tag.

2 Edit the HTML code of the toolbar title to change the displayed text (for example set it to "`Mobilizer project`").

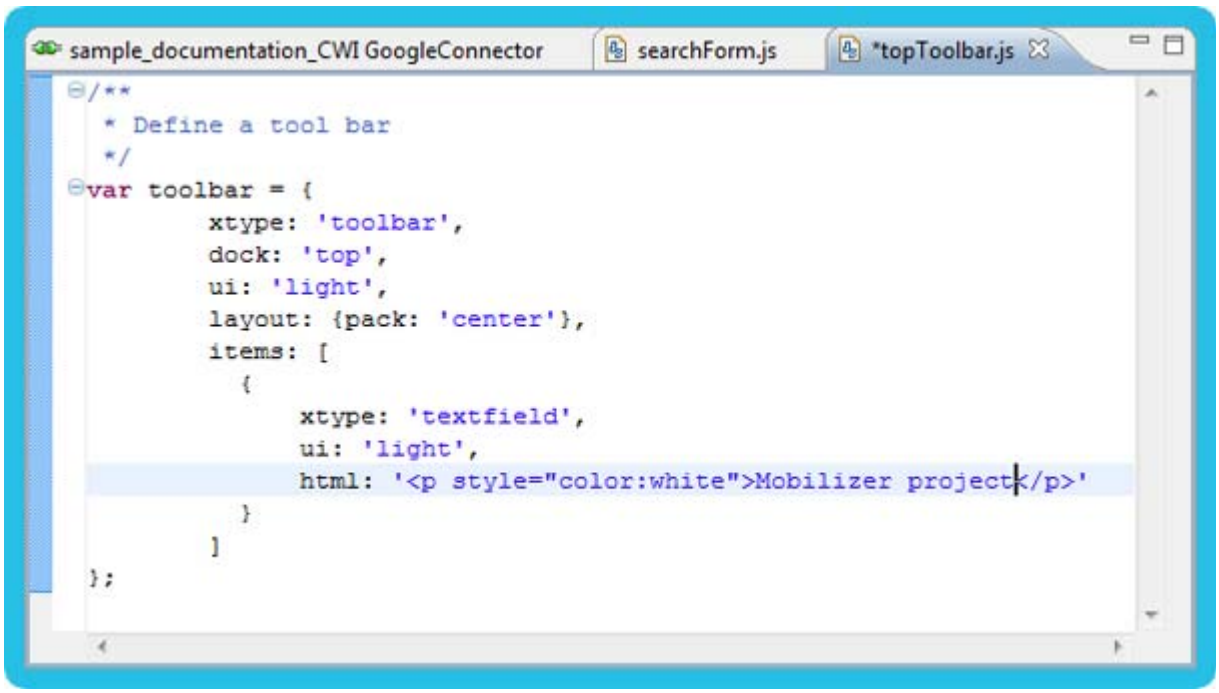


Figure 2 - 50: Updating topToolbar.js file in editor

*For detailed information about **Sencha Touch** API, refer to the Sencha Touch API documentation at: http://dev.sencha.com/deploy/touch/docs/*

3 An asterisk appears to the left of the tab file name, indicating that the file is being edited. Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

WHAT COMES NEXT?

Now that the form and the toolbar are updated to meet our application's needs, we can start testing the mobile application by displaying the search form.

2.3.3 Testing the application

The first part of the project is now set up. You can test the mobile application by calling a properly formatted URL from a Web browser to access the test platform of the project.

This section presents the test of the mobile application in the test platform included in Convertigo Mobilizer. It starts with a quick presentation of the test platform of the project where we will test in a second phase the developed mobile application.

- Test Platform presentation
- Executing the application from the Test Platform

TEST PLATFORM PRESENTATION

This section describes the test environment of the Mobilizer project, that allows executing the mobile application using any WebKit-based Web browser.

To open the test platform of the project

1 Launch a standard WebKit-based Web browser (Google Chrome for example).

2 In the URL address field, type in the URL in the following format:

```
http://<ConvertigoServer>/<ConvertigoAppName>/projects/
<ProjectName>
```

For example, in our project:

- *ConvertigoServer* = `localhost:18080` (`18080` is the default port of Convertigo Server embedded in the Studio)
- *ConvertigoAppName* = `convertigo` (default Convertigo application name in Studio configuration)
- *ProjectName* = `sampleMobilizerSencha`

It gives, in a Google Chrome URL address bar:

Figure 2 - 51: Convertigo test platform URL

Be sure to use a WebKit-based browser, such as Google Chrome.

The test platform opens:

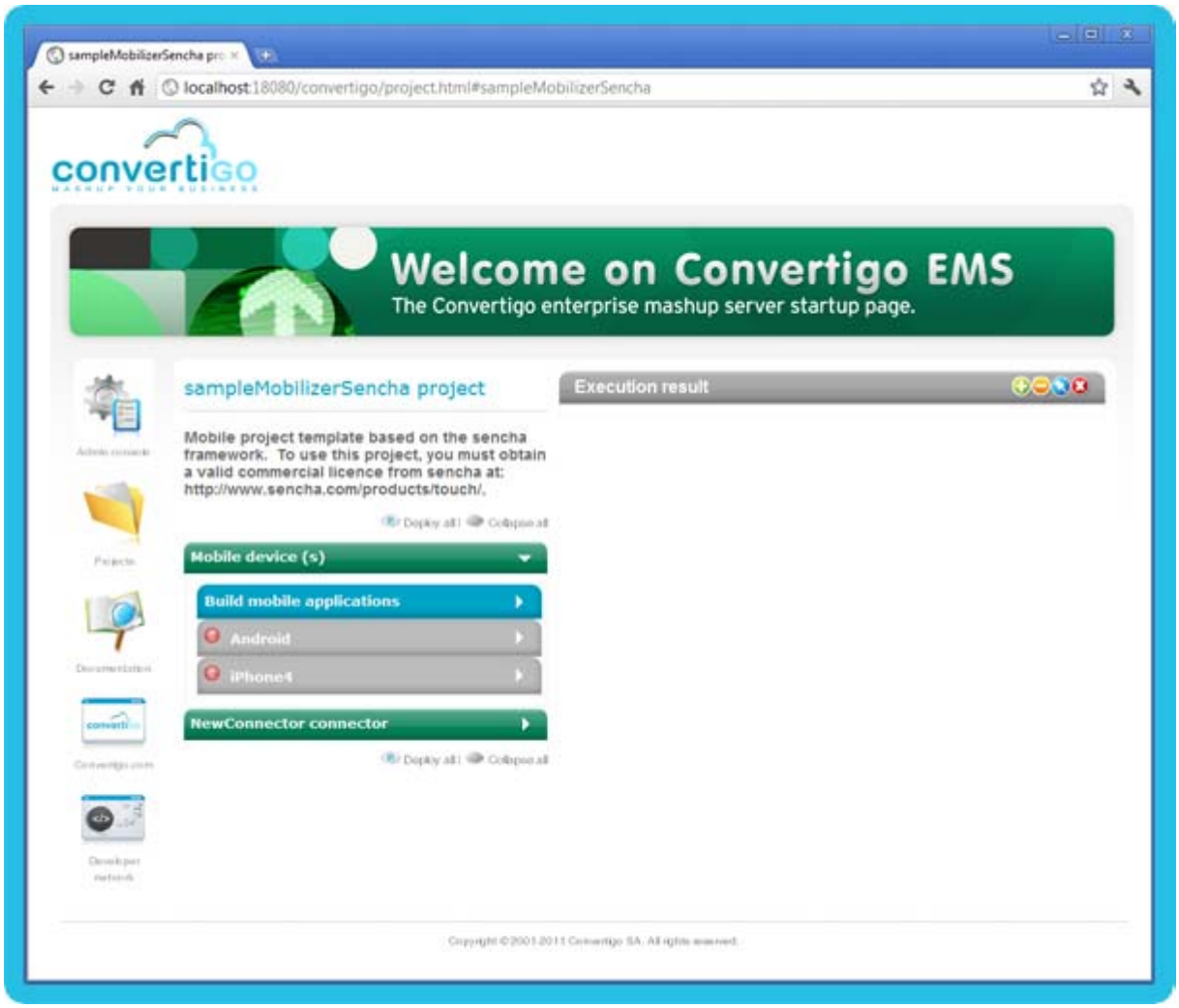


Figure 2 - 52: Test platform for the sampleMobilizerSencha project

Three columns appear on the test platform page:

- a **Convertigo toolbar**, on the left of the page;
- the **Project elements** presentation, on the middle of the page;
- the **Execution result** panel, on the right of the page.

Let's present these columns.

CONVERTIGO TOOLBAR

This toolbar contains several items allowing to access to Convertigo Server administration pages and several Convertigo Web sites providing help and documentation.

PROJECT'S ELEMENTS

The central part of the page displays the following elements of the Convertigo project:

- the **project name** (in our example, `sampleMobilizerSencha`);
- the **connectors** defined for the project (in our example, automatically created connector, named `NewConnector`);
- all **transactions** defined for each connector and their *variables* (in our example, the default transaction, named `default_transaction`, has no variable);

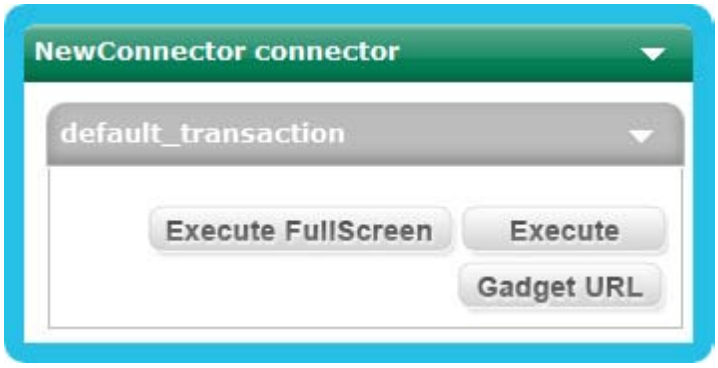


Figure 2 - 53: NewConnector and default_transaction with no variable in test platform

If several connectors are defined in the project, the test platform displays as many connector tables as defined connectors.

- the **sequences** defined for the project and their *variables* (we have no sequence in our example);
- the **mobile devices** defined for the project (in our example, `iPhone4` and `Android`).

EXECUTION RESULT PANEL

The **Execution result** panel, on the right of the page, is the panel where the mobile application runs when testing.

Figure 2 - 54: Empty Execution result panel in test platform

Let's continue with the presentation of **Mobile device(s)** table, specific to Mobilizer projects. This table contains two types of elements: the **Build mobile applications** panel and the **mobile devices** panels.

1 Expand a mobile device panel (for example `Android`):

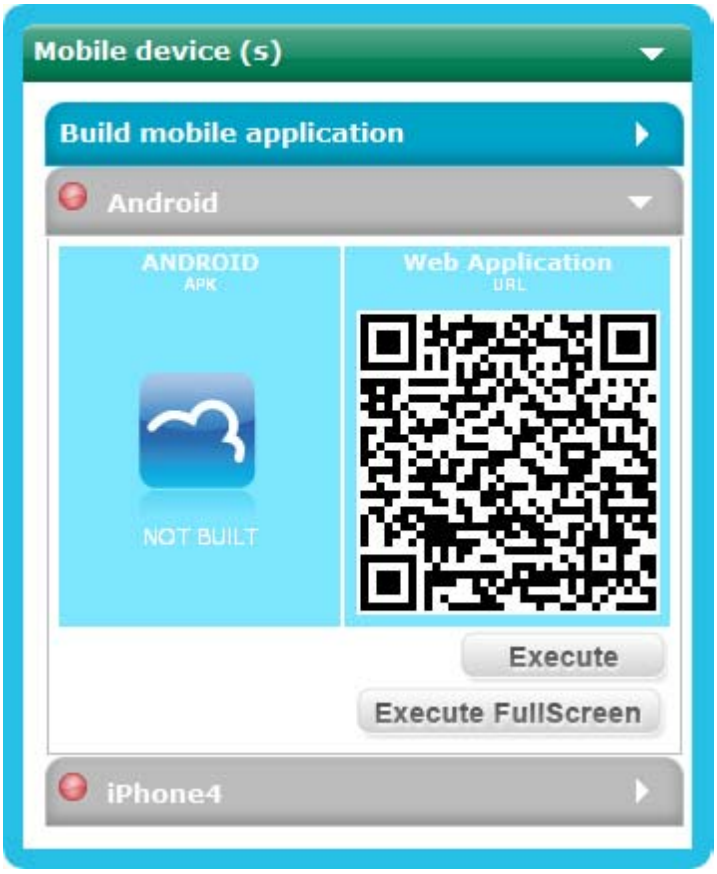


Figure 2 - 55: Android mobile device in test platform

This panel presents several elements relative to the application:

- on the left, a place where is proposed a direct access to the built application for chosen mobile device (`Android APK` in this example), available only after application build;
- on the right, the **QR code** (Quick Response code) of the web mobile application, encoding the Web application URL. You will be able to flash this QR code with your smartphone and directly execute the application on your smartphone's WebKit-based browser.
- the **Execute** and **Execute FullScreen** buttons that allow running the mobile application in current browser,
  - in the chosen mobile device (`Android` in this example) in the **Execution result** panel (with **Execute** button),
  - or directly in the full Web browser (with **Execute FullScreen** button).

2 Expand the **Build mobile applications** panel:

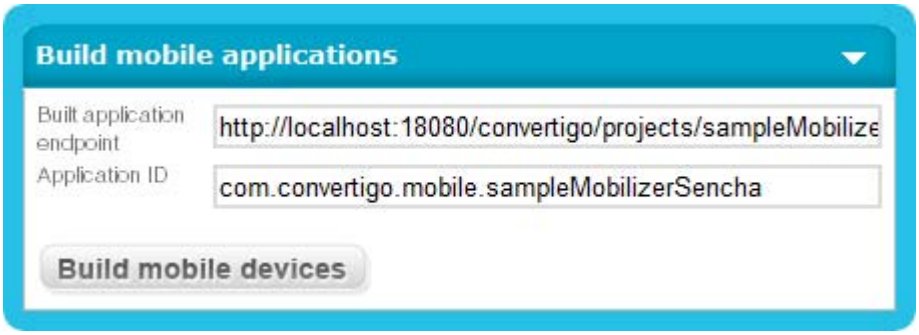


Figure 2 - 56: Build mobile applications panel in test platform

This panel can be used to build the mobile applications to be installed directly on devices. It contains a form with two input fields:

- **Built application endpoint**: once the built mobile application is installed on the mobile device, it contains the application presentation pages and JavaScript files. But, the Convertigo project(s) run(s) on a Convertigo server that has to be accessible on the Internet for the mobile application to be able from any smartphones to retrieve dynamic data. This input allows to customize the server endpoint (URL of the default Convertigo project in the Convertigo Server) to be embedded in the built application for it to be able to connect to the web services offering data.
- **Application ID**: a default package name, using the Convertigo Mobilizer project name, is generated as application ID for ***PhoneGap build***. As ***PhoneGap build*** includes limitations about special characters, such as '_', the application ID is customizable thanks to this input field if the default one based on the project name contains forbidden characters.

Finally, the **Build mobile devices** button allows to run the application build on ***PhoneGap*** in order to generate the native applications for each chosen mobile device. At the end of the build, the QR code of each generated device-specific application is displayed on the left of the device's panel:

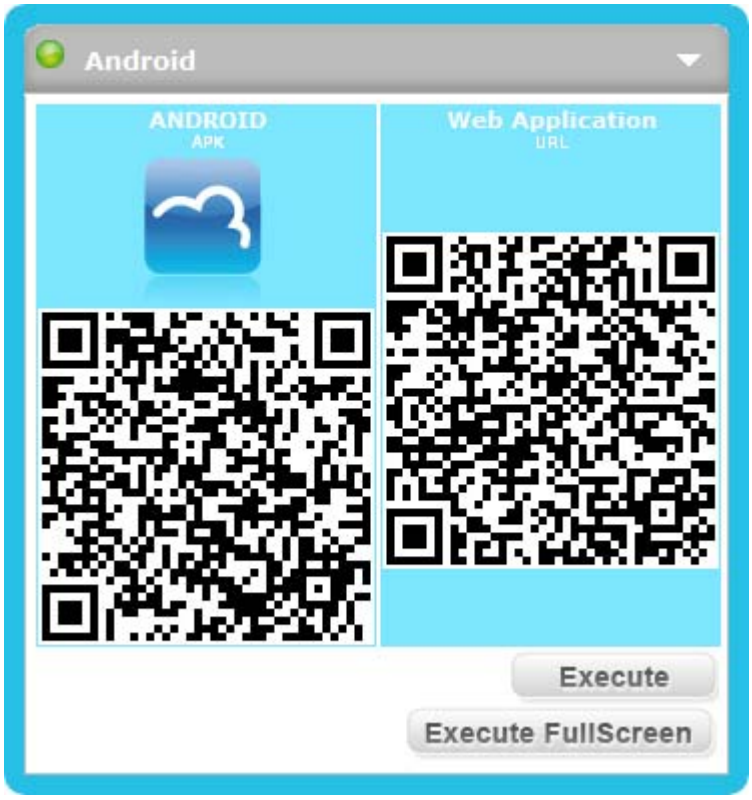


Figure 2 - 57: Built application for Android device in test platform

You are then able to flash this QR code with your smartphone's camera to download and install the application directly on your mobile (`Android` smarthphone in this example).

WHAT COMES NEXT?

Now that you are familiar with the test platform environment, you can start effectively testing the mobile application's search form, developed in the context of the `sampleMobilizerSencha` project.

EXECUTING THE APPLICATION FROM THE TEST PLATFORM

This section describes how to execute a mobile application of a Mobilizer project from the test platform included in Convertigo Mobilizer using any WebKit-based Web browser.

To execute the application using the test platform

1 Launch the test platform of the project following the procedure *"To open the test platform of the project"* on page 2 - 45.

Be sure to use a WebKit-based browser, such as Google Chrome.

2 Click on the **Execute** button of the chosen mobile device (`Android` in this example).

The **Execution result** panel displays the mobile application with the form page:

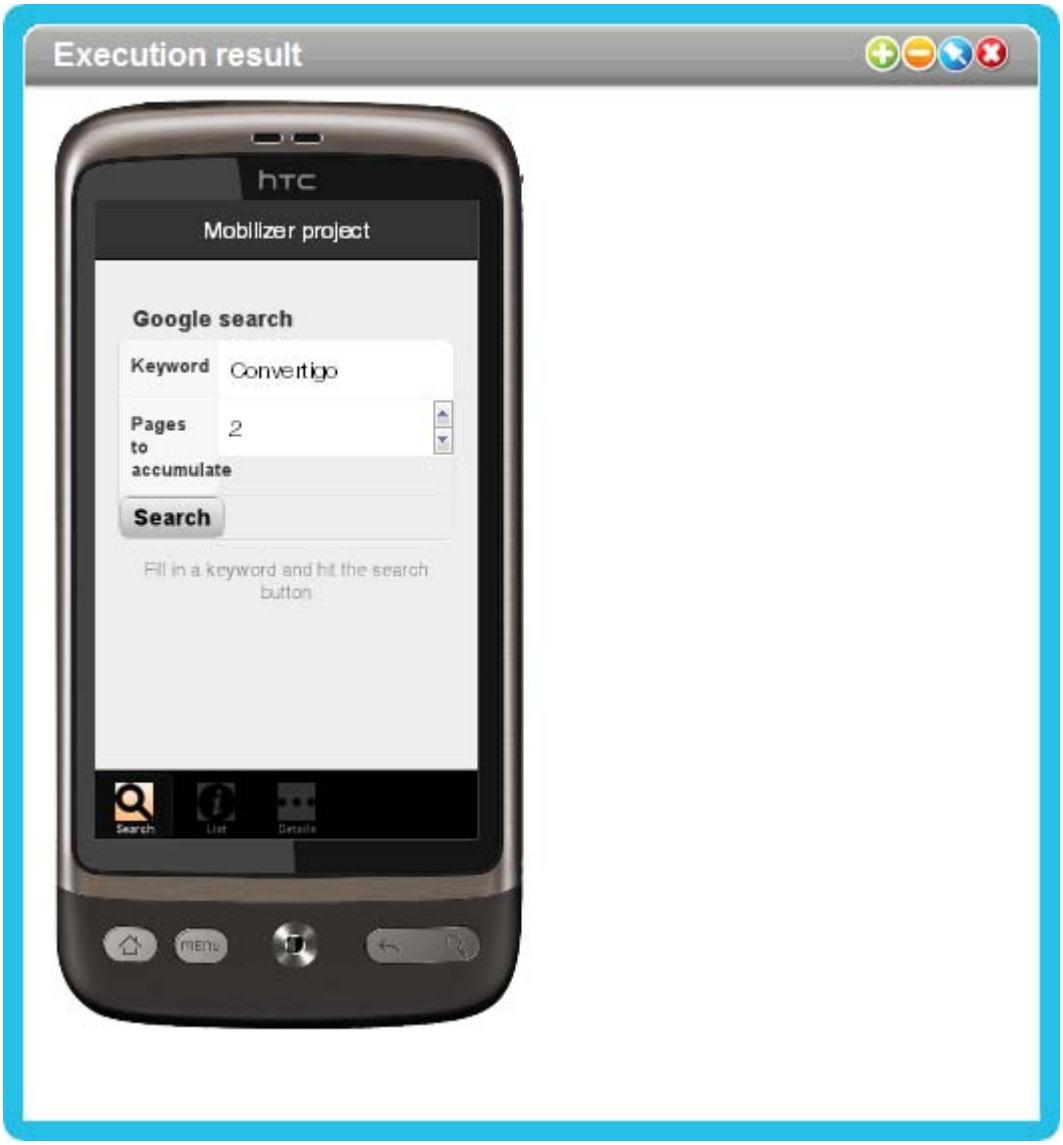


Figure 2 - 58: Application running in Android mobile device in test platform

You can notice the title of the toolbar, that corresponds to the source code updates we performed.

In the **Execution result** panel, you can use or to zoom in or out of the mobile device displaying execution result.

*You can test the mobile application on any mobile device that appears on mobile devices list of the test platform. The render will be adapted to the mobile device's size and look and feel (in the case of **auto** look and feel mode selected).*

3 You can change the default values of *Keyword* text input field and *Max pages* number field in the form:

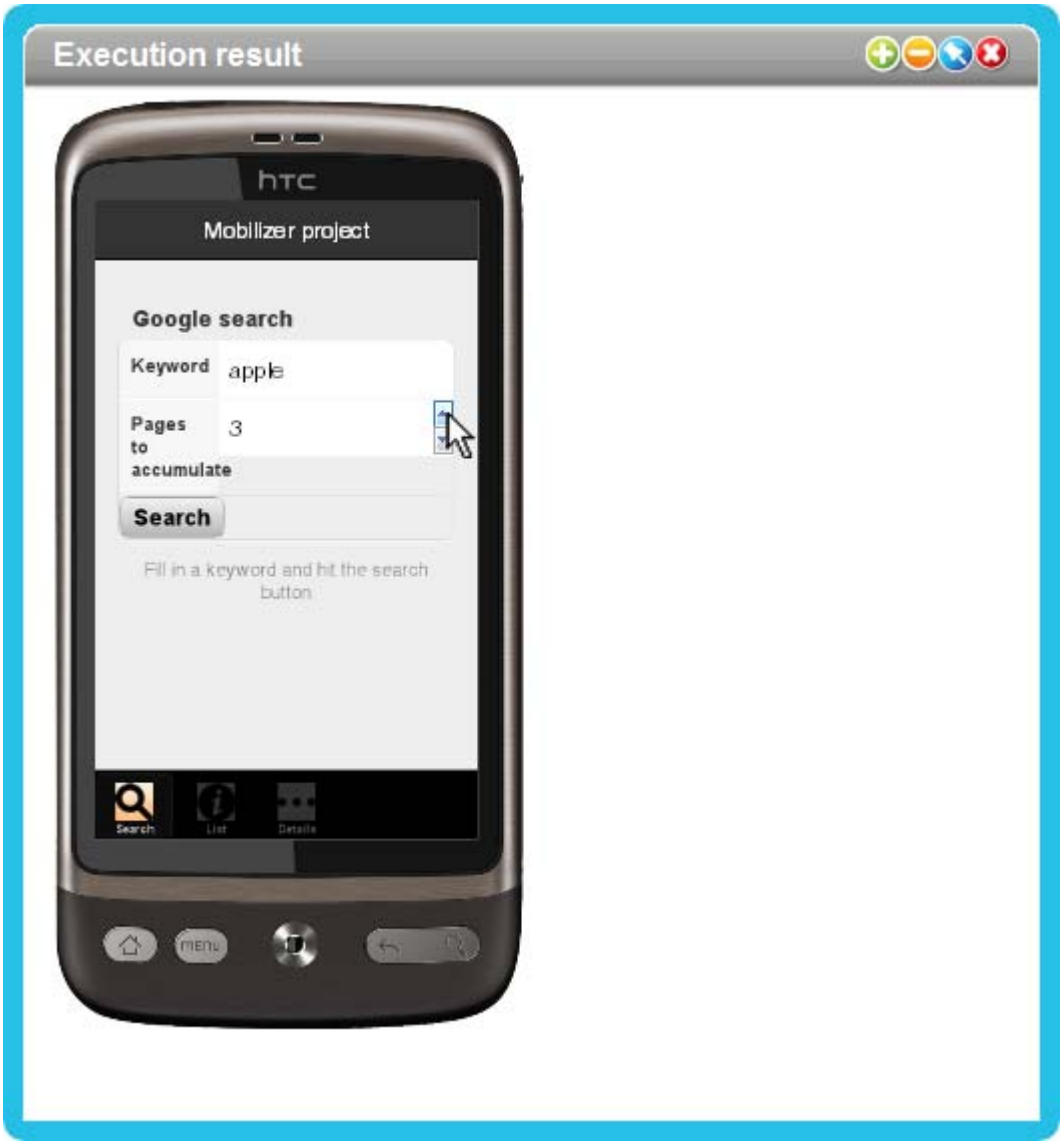


Figure 2 - 59: Changing form values in application search form

4 And click on the **Search** button to test its reaction:

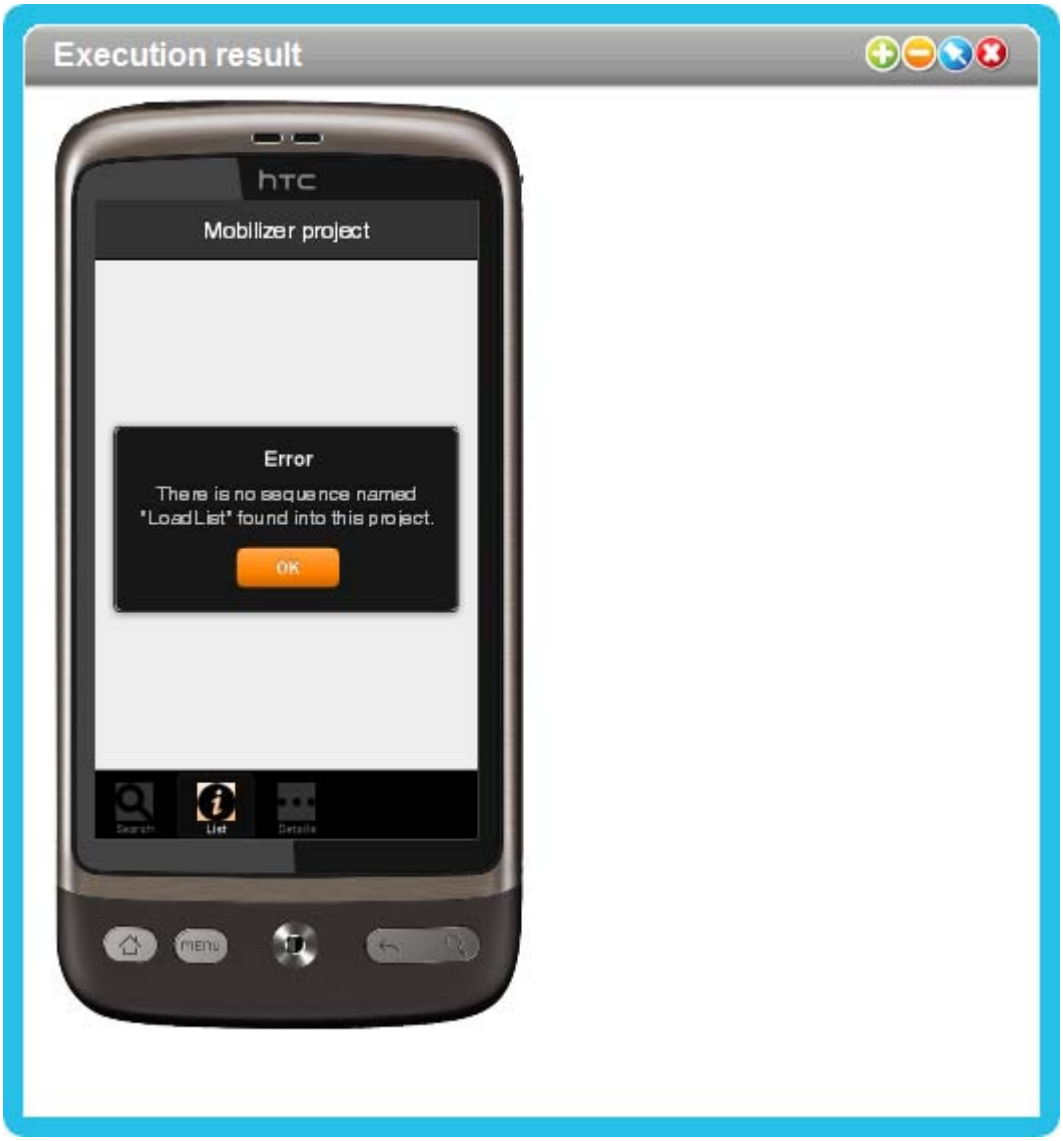


Figure 2 - 60: Error popup after clicking Search button

The list panel is activated and an error appears in a popup: the list source code has not been updated yet and tries to execute the `LoadList` sequence that was removed when cleaning the project, at the begining of this tutorial. Default source code handles the Convertigo `Exception` that is thrown and displays the error message in a popup.

You can also execute the application on the full browser page instead of in a mobile device in the **Execution result** panel.

5 To do so, click on the **Execute FullScreen** button (possibly pressing `Ctrl` key to open in a new browser tab).

The browser displays the form page of the mobile application in a new tab:

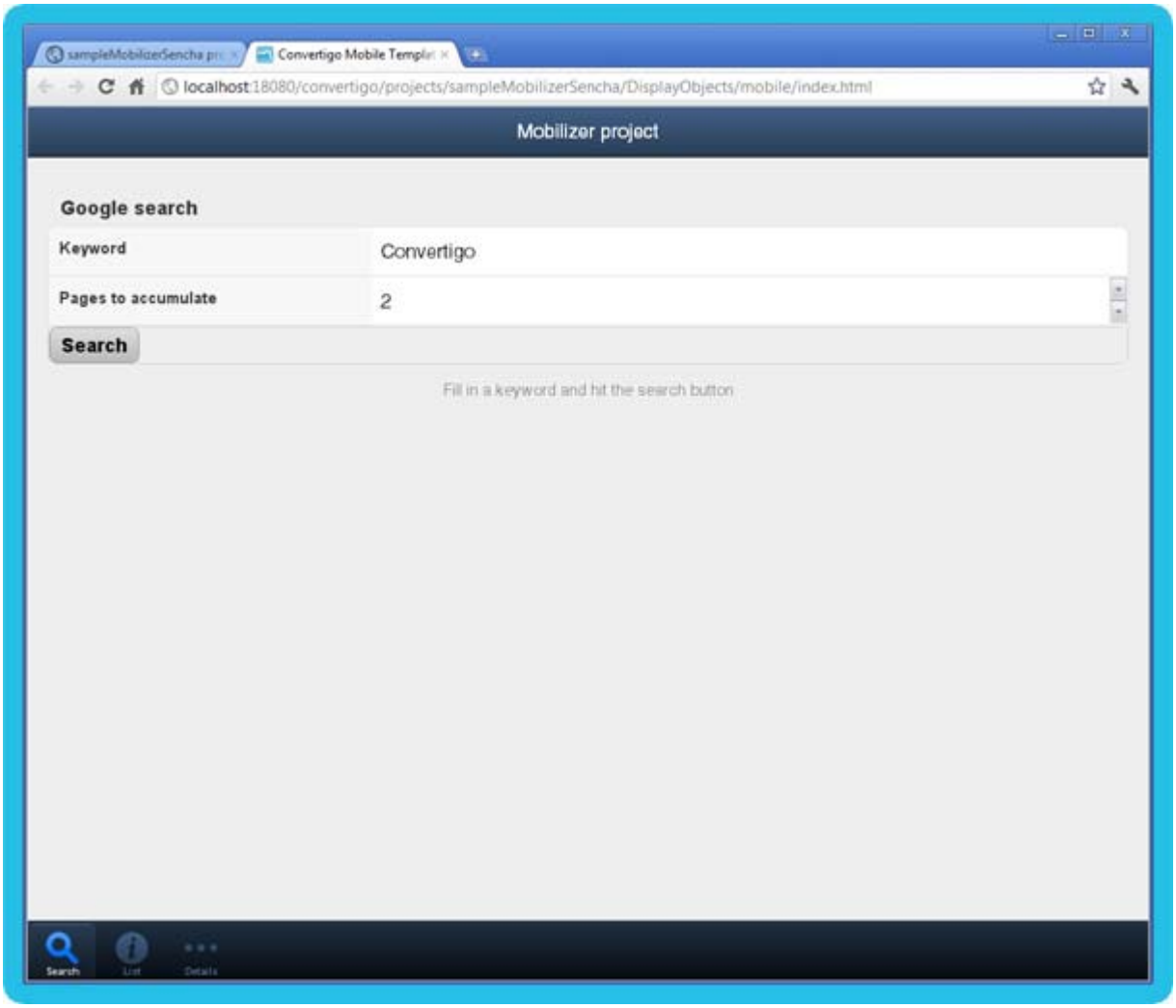


Figure 2 - 61: Application running in Web browser in test platform

You can observe that in full browser page, the mobile application look and feel is different from executing it in Android mobile device. The render is automatically switched to Sencha specific look and feel when running the application in a desktop Web browser.

WHAT COMES NEXT?

Now that we validated that the search form and the toolbar are meeting our application's needs, we have to update the list panel with the Google research, i.e. connecting the mobile application to the web service providing dynamic data.

2.3.4 Integrating dynamic data to the application

After testing the search form page of the mobile application, let's continue the development with the integration of the first web service providing dynamic data to our application.

On the search form's submit (click on the **Search** button), the application activates the list panel. This panel should display Google results retrieved from a research performed with parameters set in search form.

The list panel has to call the `searchGoogleWithLimit` transaction of the `sample_documentation_CWI` project using the `Keyword` text field and `Max pages` combo box defined in the search form as transaction's variables.

- Preparing the target Convertigo Web Service
- Integrating a list with a call to Convertigo and its response
- Executing the application from the Test Platform

PREPARING THE TARGET CONVERTIGO WEB SERVICE

Switch back to Convertigo Studio.

Before any operation, we have to make sure that the connector of the `sample_documentation_CWI` project is opened.

If the `GoogleConnector` of the CWI project is not opened in a **Connector editor**, follow the procedure *"To open a connector in its editor"* on page 2 - 55.

If the `GoogleConnector` of the CWI project is already opened in a **Connector editor**, continue with next procedure *"To run a transaction's test case and see its result in Studio"* on page 2 - 56.

To open a connector in its editor

1 In the **Projects** view, expand the web service project (for example `sample_documentation_CWI`) and then expand its `Connectors` folder.

2 Double-click on the Web Integrator connector (for example `GoogleConnector`) to open it in its **Connector editor**:

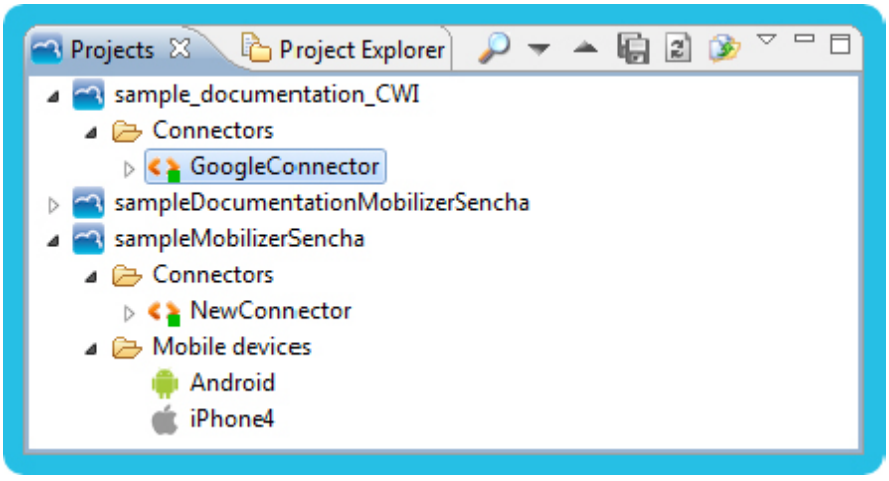


Figure 2 - 62: Opening CWI connector in editor

The connector opens in a new tab (for example `sample_documentation_CWI GoogleConnector`). Convertigo automatically connects to the HTML page defined as connector root (for example `www.google.com`).

This page is diplayed in the **Web Browser**:

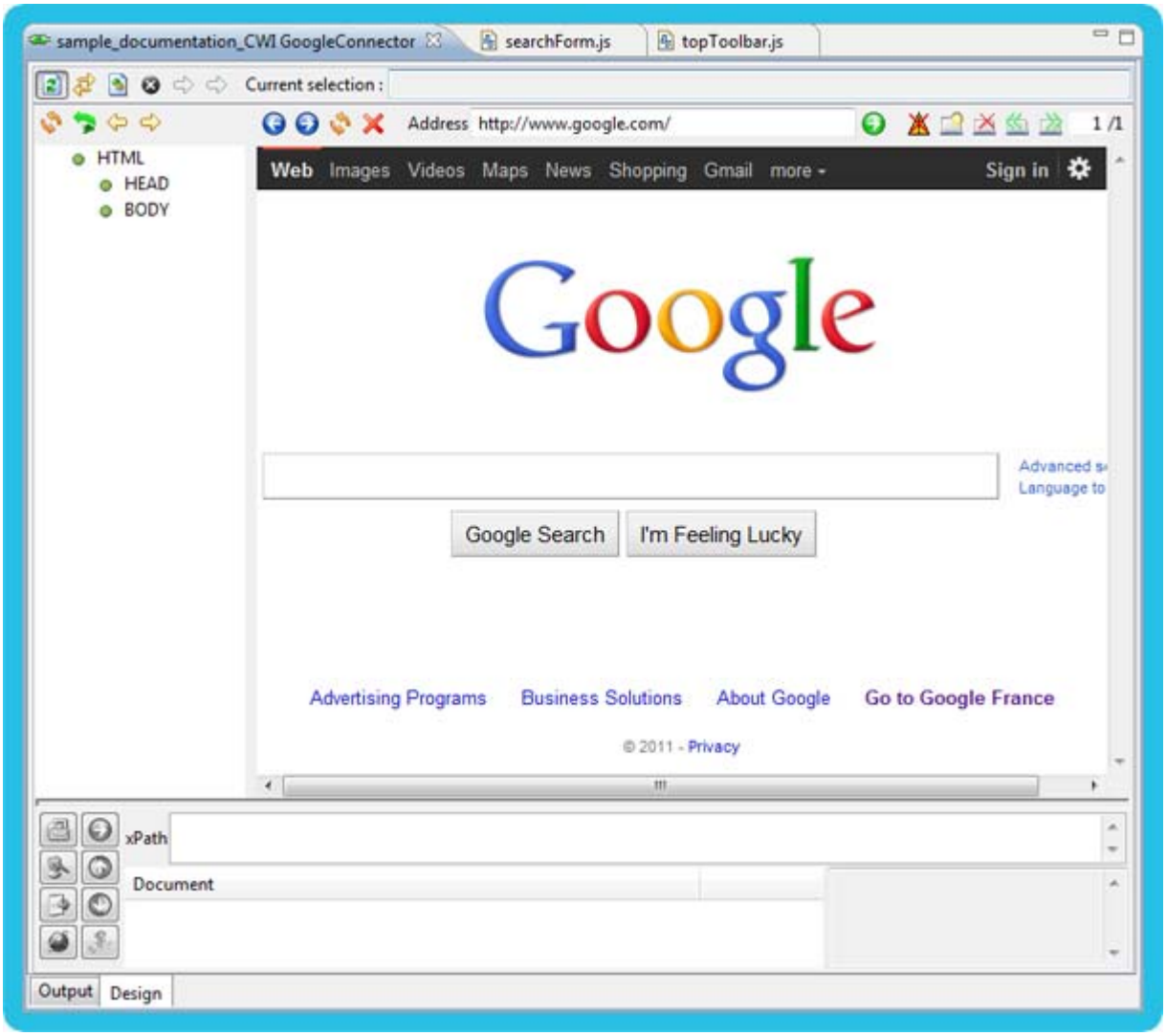


Figure 2 - 63: CWI connector editor (DOM Tree, Web Browser, XPath Evaluator)

WHAT COMES NEXT?

Now that the target Web Integrator connector is opened in its editor, we can run one of the transaction's test cases in order to remember its correct response schema.

To run a transaction's test case and see its result in Studio

1 In the **Projects** view, expand the `GoogleConnector` connector, then **Transactions** folder, `searchGoogleWithLimit` transaction, and finally **Test cases** folder.

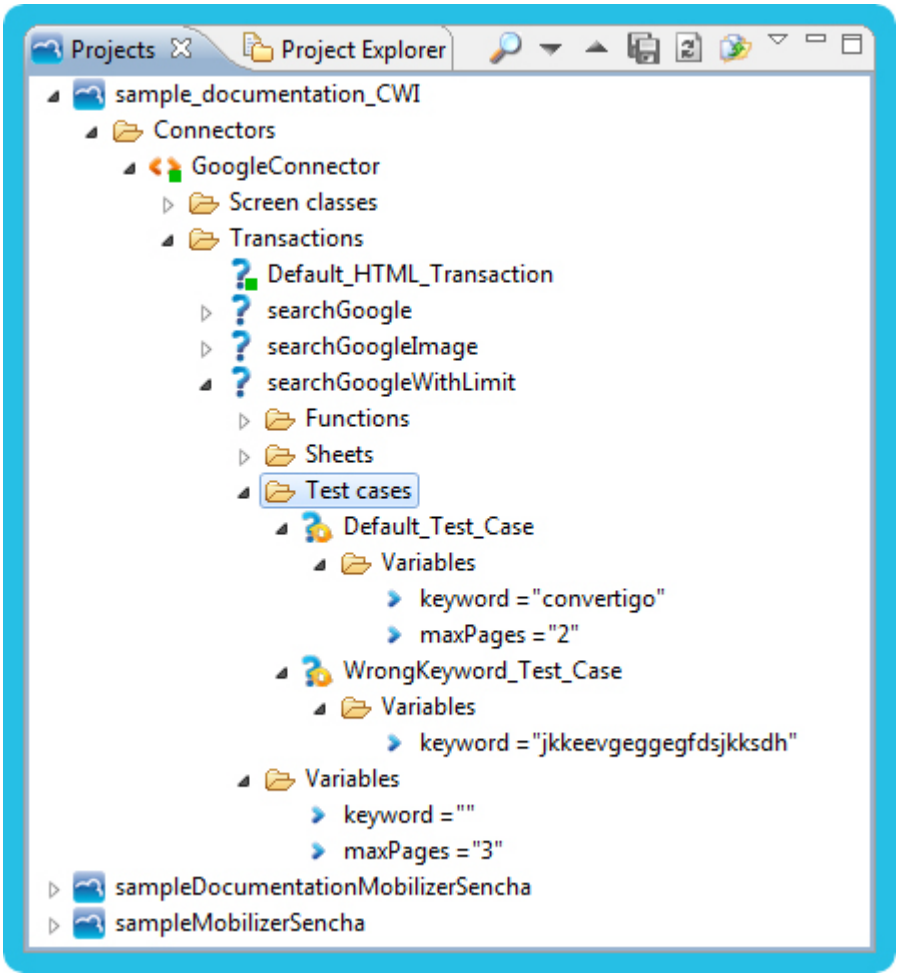


Figure 2 - 64: Available test cases of searchGoogleWithLimit transaction

2 Right-click on the test case, for example `Default_Test_Case`, and select **Run**.

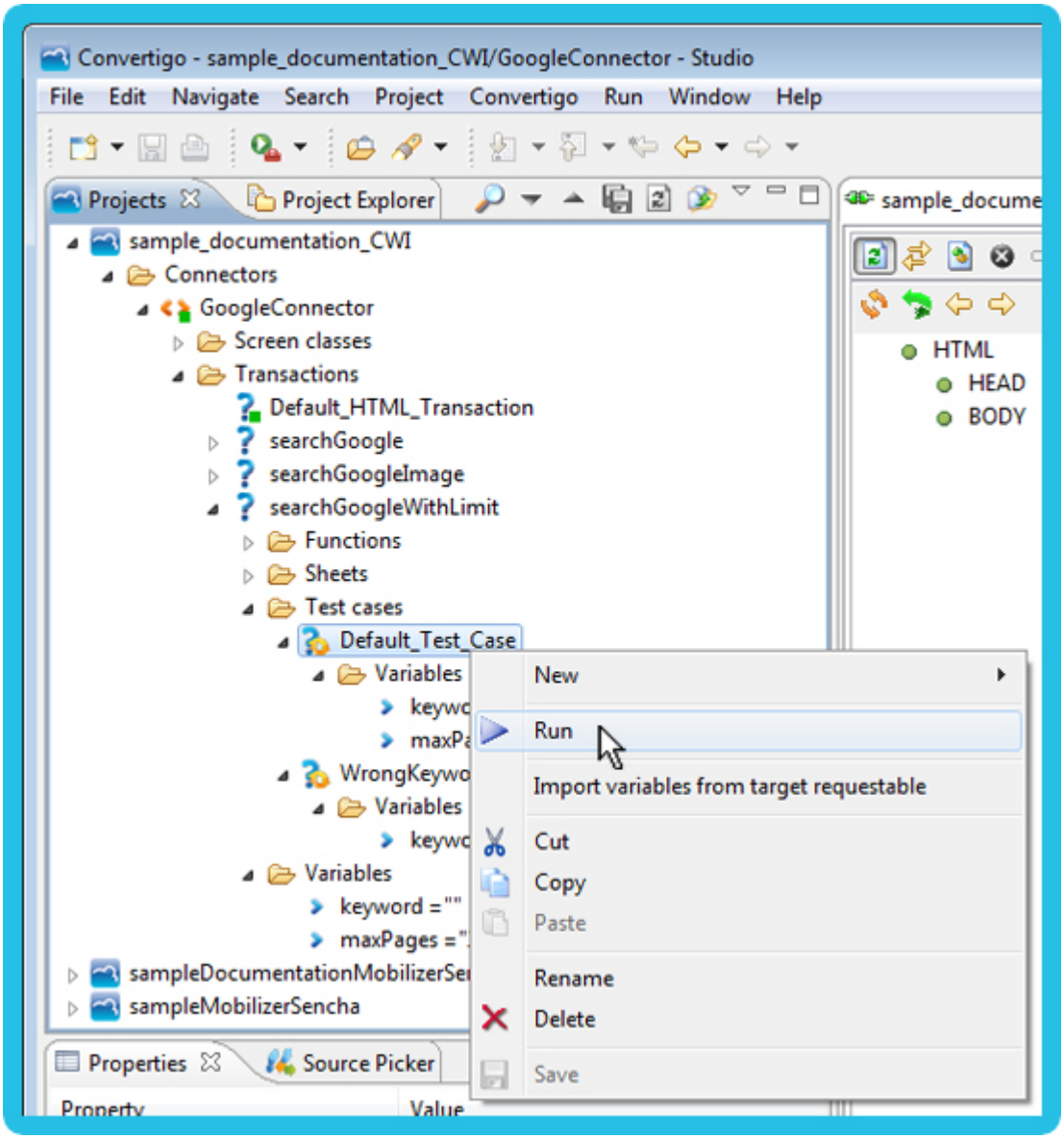


Figure 2 - 65: Running test case in Convertigo Studio

The transaction is executed in the connector opened in its editor. The transaction stop button is active in the **Connector editor** ( ).

3 When the transaction ends, the stop transaction button is not active anymore in the **Connector editor** ( ), click on the **Output** tab of the editor.

The transaction result is displayed in the XML tab:

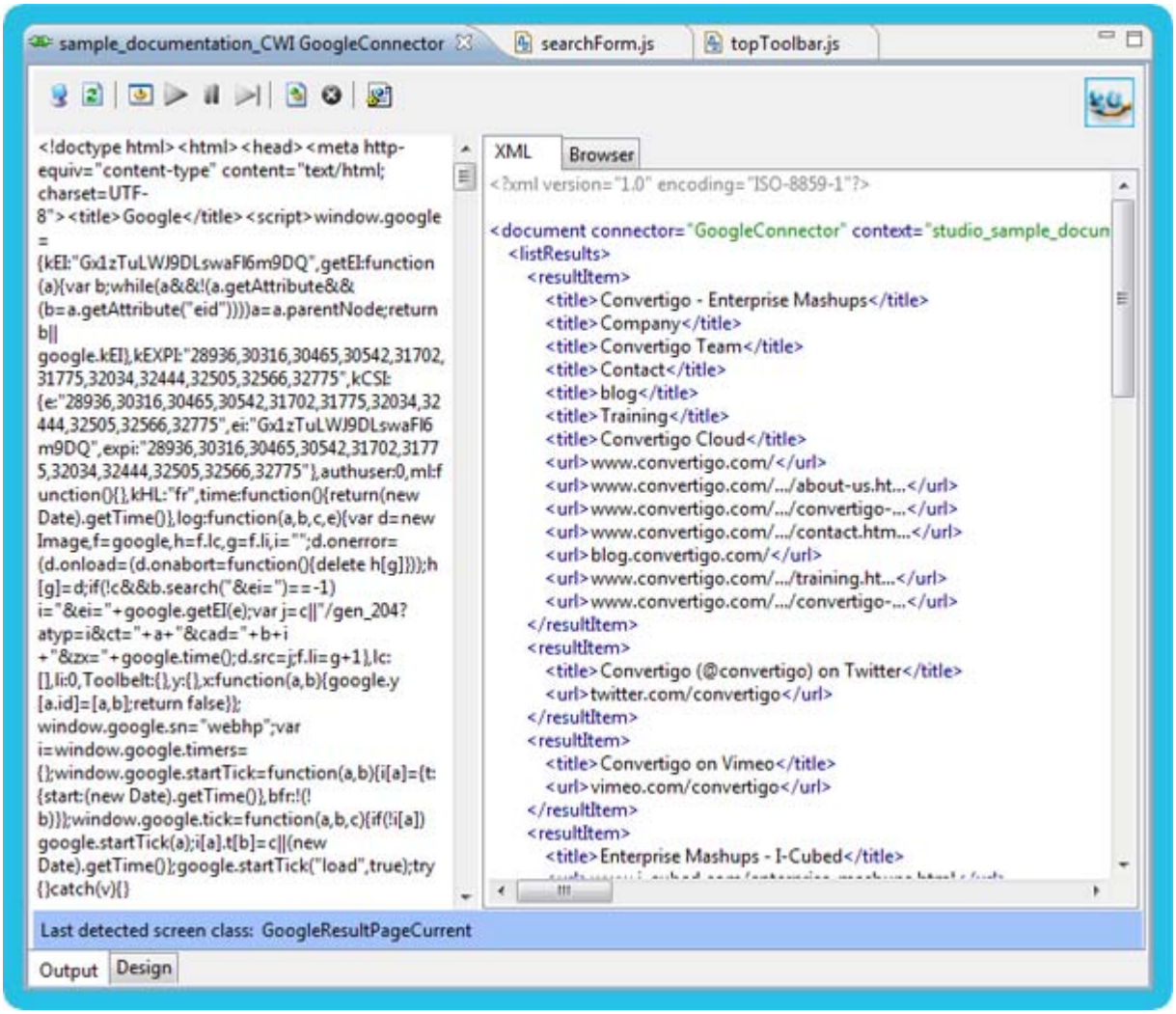


Figure 2 - 66: Transaction result displayed in Output tab of Connector editor

WHAT COMES NEXT?

Now that the target transaction is operational and that we know its correct response schema, we can define the call to the `searchGoogleWithLimit` transaction and its response display in the list when list panel is activated.

INTEGRATING A LIST WITH A CALL TO CONVERTIGO AND ITS RESPONSE

This section describes how to integrate the response of a call to a Convertigo transaction (or sequence) with a ***Sencha Touch*** list display object.

To create a list displaying the results of a Convertigo transaction

In previously selected features, the *List display* feature was chosen. A JavaScript file was created in the project environment to implement this page: `listDisplay.js`.

1 In the **Project Explorer** view, expand the `sources` folder and double-click on the `listDisplay.js` file to open it in the editor.

Default source code found in this file corresponds to the basic Mobilizer template application. It is the result list of room booking form panel example.

Before the list panel itself, the source code contains two objects:

- the declaration of a data model, that defines the data structure corresponding to the columns of the retrieved data, and which is used by the `Store` to store the response data.
- the creation of a `C80 store` object, that keeps the records of the transaction (or sequence) result after its execution, using the previously defined data model and defining a root element. This root is the iterated element from the response data that contains the columns described in the model.

Then the panel is implemented with:

- a title,
- the inclusion of the top toolbar,
- a dock icon,
- a list object defining:
  - the list item template that describes how to display each line of the list thanks to an HTML piece of code including the columns of the model,
  - the store that the list uses to retrieve the transaction (or sequence) response data, it points here on previously defined `C80 store` object,
  - the actions to execute on item disclosure, which is the source code to execute when clicking on the arrow icon displayed on the right of each list item,
- and the code to execute when activating the panel, which is the code that calls Convertigo transaction and loads the response data in the list.

*`C80 Store` object, offered by **Convertigo mobilelib** library, is derived from **Sencha Touch** `Store` object. It allows calling a Convertigo transaction or sequence and keeps result records after its execution.*
For detailed information on the `Store` object, with all its properties and methods, refer to the Sencha Touch API documentation at: http://dev.sencha.com/deploy/touch/docs/?class=Ext.data.Store

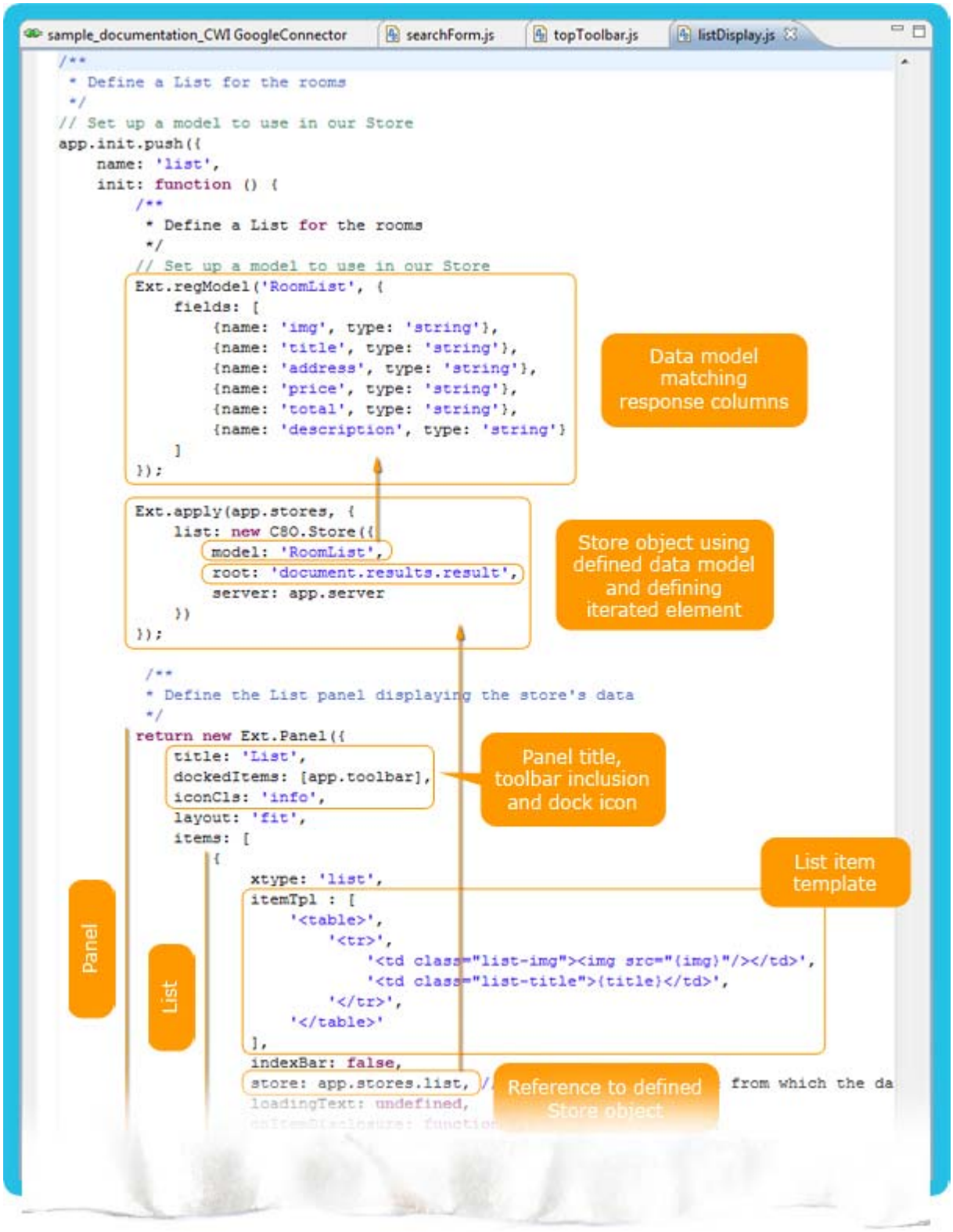


Figure 2 - 67: Default listDisplay.js file displayed in editor - part 1

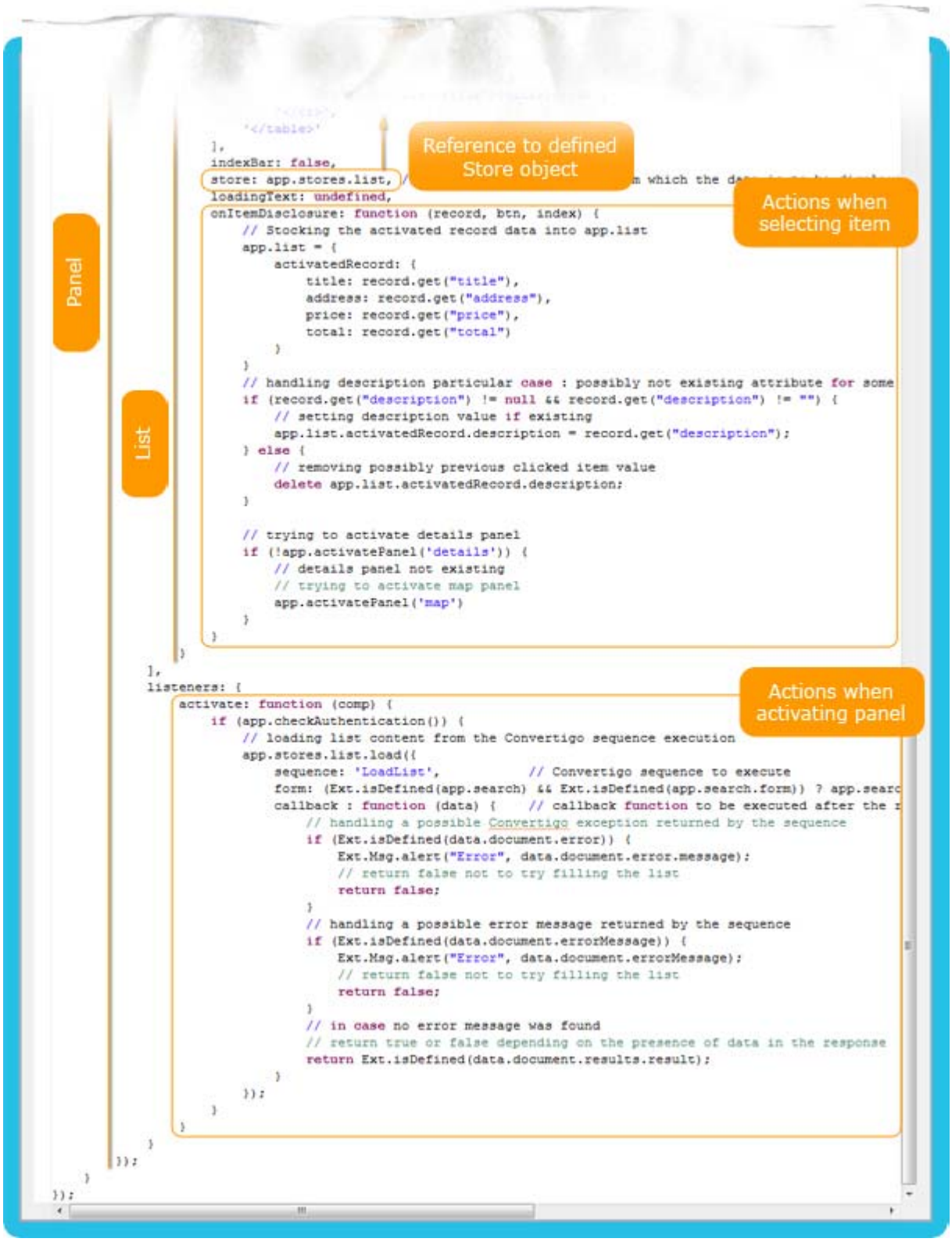
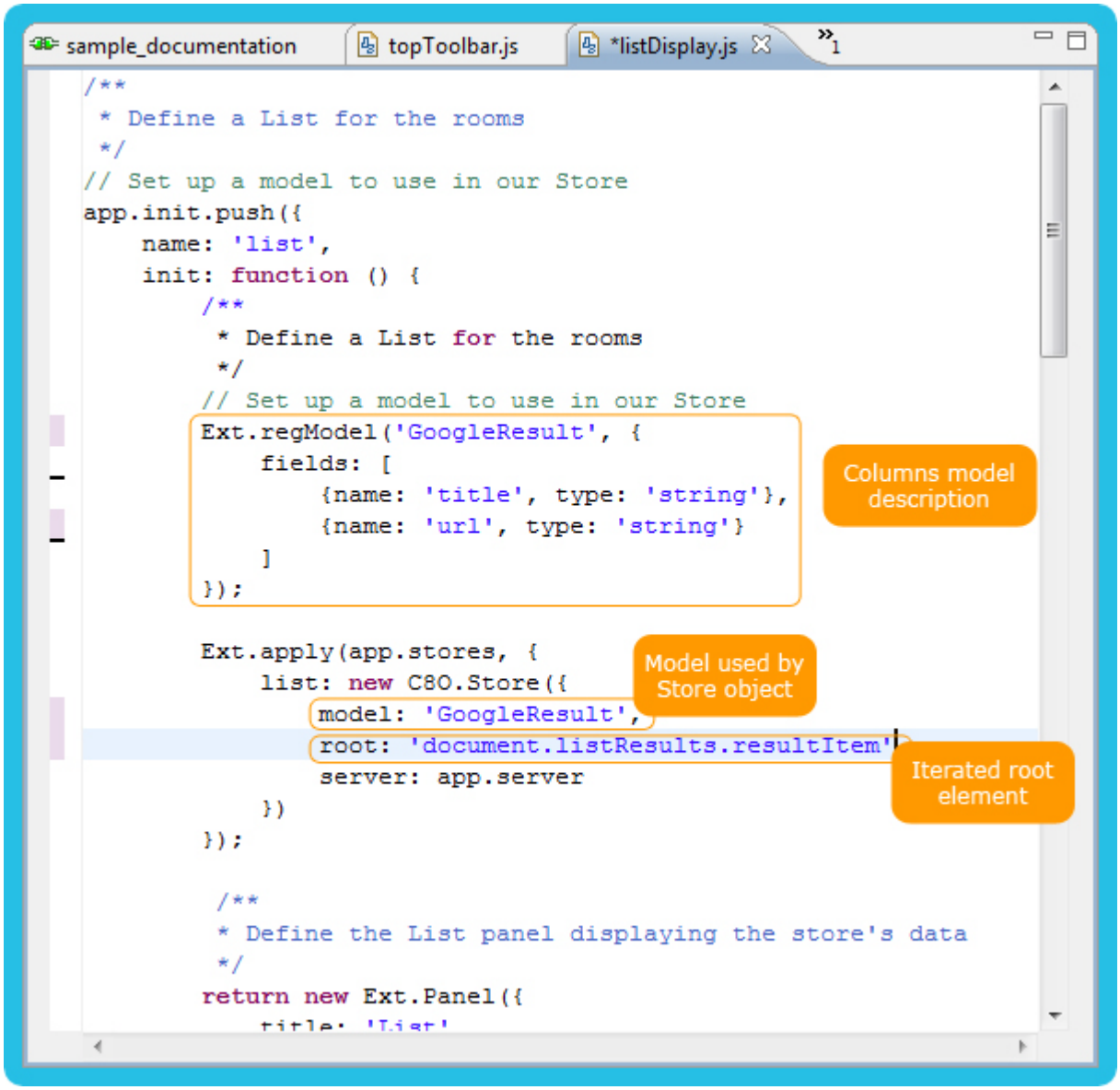


Figure 2 - 68: Default listDisplay.js file displayed in editor - part 2

The default source code has to be modified according to the needs of the currently developed application.

In a first time, we have to modify the model defintion as well as the root element of the store object. These two lines of code should match the transaction's response schema.

2 Modify the code up to get the following source code:




```
/**
 * Define a List for the rooms
 */
// Set up a model to use in our Store
app.init.push({
    name: 'list',
    init: function () {
        /**
         * Define a List for the rooms
         */
        // Set up a model to use in our Store
        Ext.regModel('GoogleResult', {
            fields: [
                {name: 'title', type: 'string'},
                {name: 'url', type: 'string'}
            ]
        });

        Ext.apply(app.stores, {
            list: new C8O.Store({
                model: 'GoogleResult',
                root: 'document.listResults.resultItem',
                server: app.server
            })
        });

        /**
         * Define the List panel displaying the store's data
         */
        return new Ext.Panel({
            title: 'List'
```

Figure 2 - 69: Updating listDisplay.js file in editor - part 1

You can find the complete edited `listDisplay.js` file in the example project `sampleDocumentationMobilizerSencha` resources. You can copy and paste source code from it to save time.

This updated source code allows to:

- define a **new data model**, including the two columns returned by the Convertigo transaction, `title` and `url`;
- update the store object to **link** it with this new model;
- and define a **new root element** on the store object to iterate in data (you can find the correct path by observing the transaction result XML using the procedure *"To run a transaction's test case and see its result in Studio"* on page 2 - 56).

Beware! Follow the steps of the tutorial, do not copy and paste the complete source code. At this stage, the `panel` object with all its properties and functions is not yet updated.

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

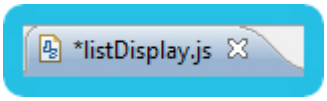


Figure 2 - 70: Edited listDisplay.js file

3 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

In a second time, we have to modify the list template, piece of HTML code that describes how to display the data on each line of the list.

4 Modify the code up to get the following source code:

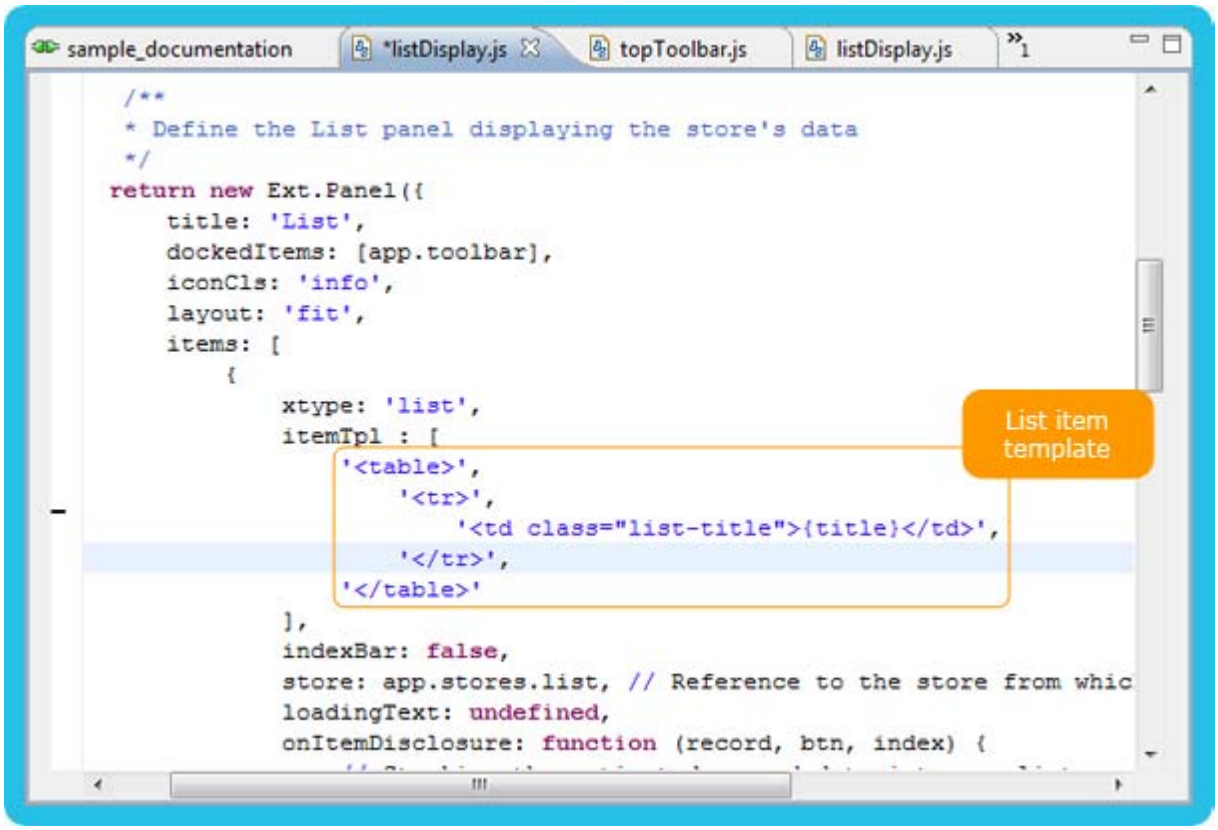


Figure 2 - 71: Updating listDisplay.js file in editor - part 2

You can find the complete edited `listDisplay.js` file in the example project `sampleDocumentationMobilizerSencha` resources. You can copy and paste source code from it to save time.

This updated source code allows to define a **new item template** that displays the `title` column of each line returned by the Convertigo transaction in an HTML table.

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

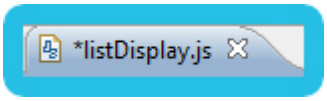


Figure 2 - 72: Edited listDisplay.js file

5 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S.`

Then, the next part of the source code is dedicated to the management of the code to execute on `disclosure` event, i.e. on the click on the arrow icon appearing on the right of a list item.

For now, we don't want to deal with that event, as this is a step that will come later in the application development.

6 Select the content of the `onItemDisclure` function and press `Ctrl+'/'` keys. This will add all lines under comments by adding '`//`' characters on the begining of each line of code:

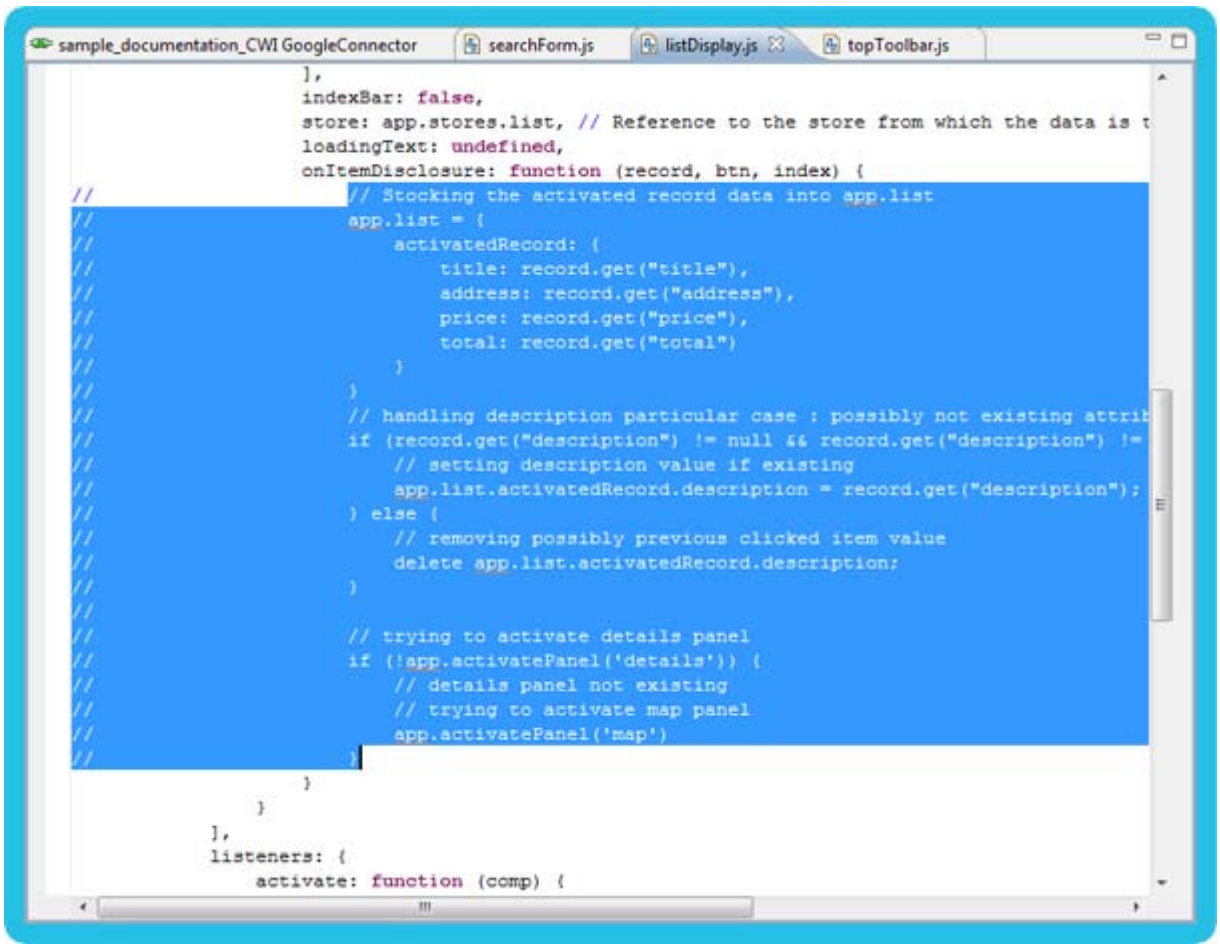


Figure 2 - 73: Updating listDisplay.js file in editor - part 3

Beware to select only the content of the function without its closing brace! Otherwise, it would generate a JavaScript exception at runtime and the list panel would not be displayed correctly.

7 Unselect the lines by clicking anywhere else in the page or in Convertigo Studio.

8 Save the updated file by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

The final part of the source code is the most important to edit. It is the part of the code that implements the call to Convertigo transaction (or sequence) on the activation of the panel (`activate` event). The template existing code already calls a Convertigo sequence, we have to update it to call the existing transaction from CWI project.

9 Modify the code up to replace the `if(app.checkAuthentication())` inner code by the following source code:

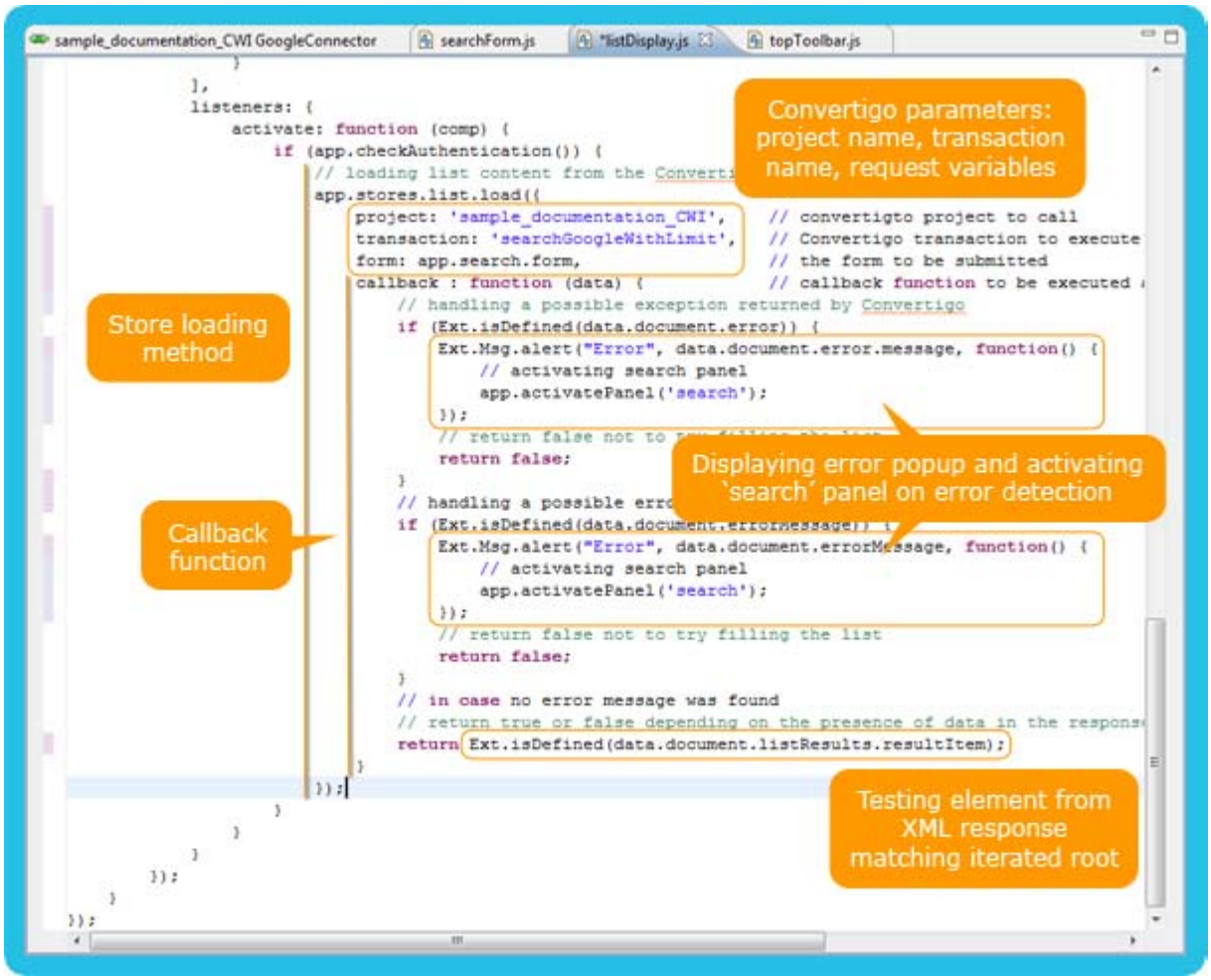


Figure 2 - 74: Updating listDisplay.js file in editor - part 4

*The `app.checkAuthentication()` method is testing that the user is well authentified in the login panel before anything else, in case the **Login form** feature has been selected in the project creation wizard.*
*In our application, the **Login form** feature is not implemented, the `app.checkAuthentication()` method will therefore always return `true`. We can keep it in source code, it has no effect.*

The modified piece of source code is the store loading method. This method is separated in two parts:

- the Convertigo parameters definition:
  - allows to target a project different than the current Mobilizer project (in our example, the CWI `sample_documentation_CWI` project);
  - then to define the name of the transaction (or sequence) to request (in our example, the `searchGoogleWithLimit` transaction), as well as the connector name, if it is not the default connector (in our case it is the default connector of the project, so it is not precised here);
  - and the variables to send with the request to Convertigo and their values (here by a form submission, that was declared in the search panel). For more information about the form, see *"To create a form page"* on page 2 - 38.
- the `callback` handler function: allows to define the actions to perform when the response is back from Convertigo.
  - Here, we simply update the error handling already implemented by the template project by activating the 'search' panel on error message popup validation.
  - This `callback` handler function **must** return a boolean value, in order to inform the list whether data to load is present or not. In cases of error, the `return false` statement is present to inform the list not to try filling with data. In case no error is found, it returns `true` or `false` depending on the presence of the iterated root element in data.

In order to avoid JavaScript Exceptions when loading a list from a `Store`, you should always keep the `return Ext.isDefined(rootElement)` statement at the end of the `callback` handler function.

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

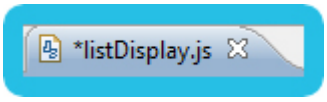


Figure 2 - 75: Edited listDisplay.js file

10 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

WHAT COMES NEXT?

When the transaction returns, the application should display the list page. Next step is to test this feature to check if it meets our application's needs.

Now that the list page is updated to meet our application's needs, we can start testing the transaction execution and its result displayed in the list.

EXECUTING THE APPLICATION FROM THE TEST PLATFORM

This section describes how to test our application again and see the results displayed in the list.

To execute the application using the test platform

1 Launch a research in the mobile application by following steps 1 to 4 of the procedure *"To execute the application using the test platform"* on page 2 - 50.

For this test, we can try executing the `iPhone4` mobile device version, as well as

keeping the default values set in the search form (but you can also modify these values).

After clicking on the **Search** button in the mobile application displayed in the **Execution result** panel, the loading mask is displayed while the transaction is executed on Convertigo:

Figure 2 - 76: Activation of loading mask after clicking Search button

This is the mask associated with the `list Store` object, that is used in `listDisplay.js` file to call Convertigo transaction and retrieve results. The mask was already declared in the `app.js` file part of the default template project's features.

For more information about `app.js file`, see "SOURCES directory" on page 2 - 28.

The mask activation allows preventing the user to perform any action on the application while loading data from Convertigo transaction execution. The mask activation and deactivation are automatically synchronized with the `Store` object and its loading process.

Once all data are loaded, the **Execution result** panel displays the mobile application with the resulting list:

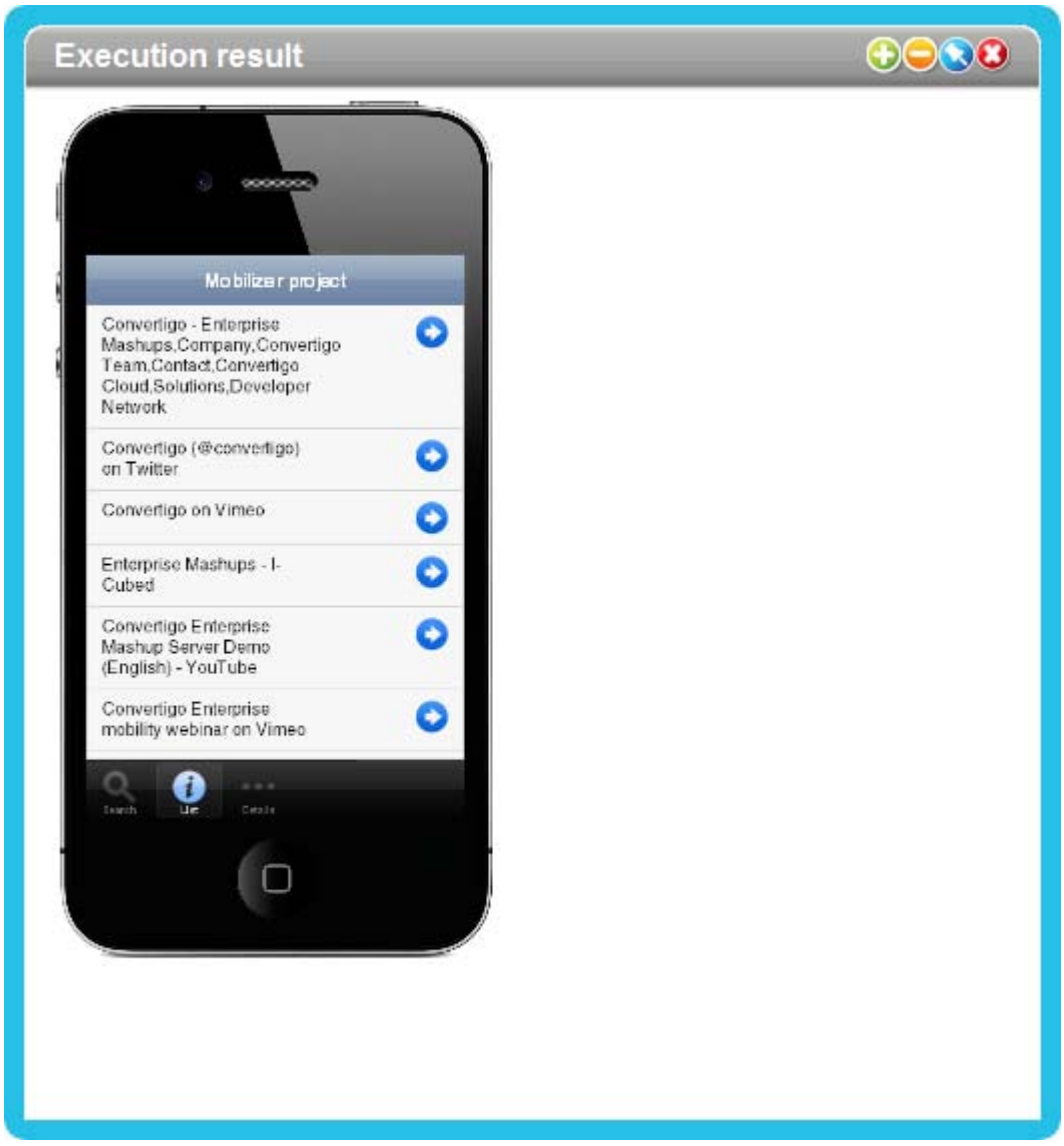


Figure 2 - 77: Result list after search perfomed in iPhone4 device in test platform

2 You can select a result item in the list, it appears selected:

Figure 2 - 78: Item selected in result list in test platform

On the right of each line, the right arrow button is added automatically by ***Sencha Touch*** framework because of the presence of the `onItemDisclosure` event handler function in the source code.

3 As code to execute on this handler has been commented in source code (refer to step 6 of procedure *"To create a list displaying the results of a Convertigo transaction"* on page 2 - 59), you can try clicking on the button: it does nothing for now.

You can update source code of the `list` object in `listDisplay.js` file to comment the `onItemDisclosure` function and its closing brace (in addition to commenting its content). This would have the effect of not displaying the right arrow button on each line of the list.

What comes next?

Now that we tested the search transaction is correctly launched with the values set in the search form and the list correctly displays the results retrieved from Convertigo transaction, we can implement the code of the `onItemDisclosure` event handler function in the source code

to continue the application's development.

2.3.5 Improving the application functionalities

After testing the list page of the mobile application, let's continue the development with the display of a details sheet when clicking on one item of the list. Then, we will dynamically complete data coming from the list with data coming from a second web service in the details sheet.

On a list item disclosure (click on the right arrow button), the application activates the details sheet panel. This panel should display the `title` and `URL` of the selected item of the list (the `URL` being a piece of data present in search results but not displayed by the list). These data are saved from the selected line when clicking on the list item button, before activating the panel.

The details sheet panel has to call the `searchGoogleImage` transaction of the `sample_documentation_CWI` project using the selected item's `title` piece of data as transaction's variable.

- Displaying a details sheet on list items disclosure
- Setting up the details sheet user interface
- Integrating a details sheet with a call to Convertigo and its response

DISPLAYING A DETAILS SHEET ON LIST ITEMS DISCLOSURE

This section explains step by step how to display a details sheet when clicking on one item of the list.

To display a details sheet on selection of a list item

This procedure completes *"To create a list displaying the results of a Convertigo transaction"* on page 2 - 59 as it will finish editing the list feature.

1 Switch back to Convertigo Studio where `listDisplay.js` file was opened for edition.

The content of the `onItemDisclure` function is commented, due to previous edition of the file (see Figure 2 - 73).

2 Uncomment the content of the event handler by following steps 6 to 7 of the procedure *"To create a list displaying the results of a Convertigo transaction"* on page 2 - 59.

The same shortcuts work in to comment or uncomment lines of code in Eclipse, and therefore in Convertigo Studio.

Default source code found in this handler function corresponds to the basic Mobilizer template application. It is the default behavior to execute when clicking on a list item right arrow button.

This behavior consists in:

- stocking the activated list item record data into a global variable in order to keep and reuse the data;

- trying to activate the details panel, and if not successful, activating the map panel.

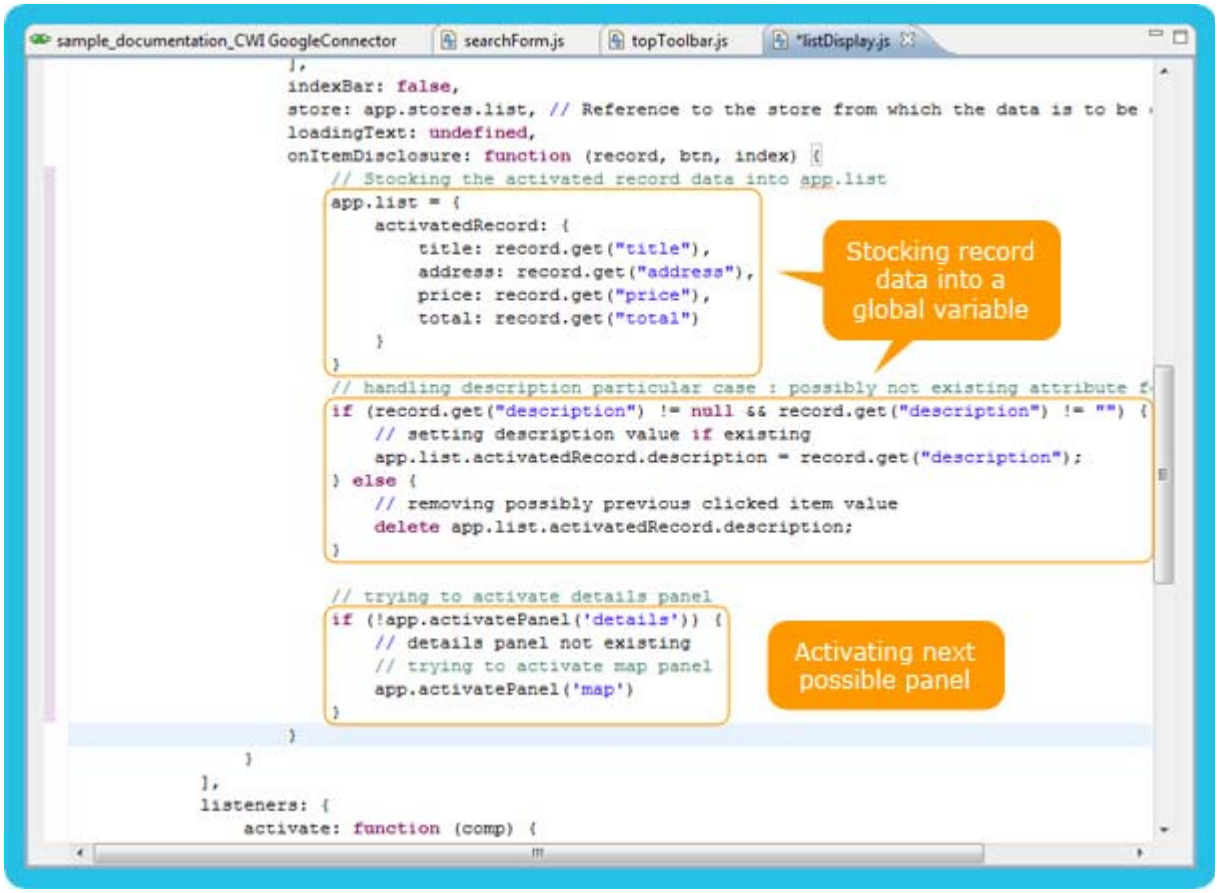


Figure 2 - 79: Default behavior on list item activation in listDisplay.js file

The default source code has to be modified according to the needs of the currently developed application.

3 Modify the code up to get the following source code:

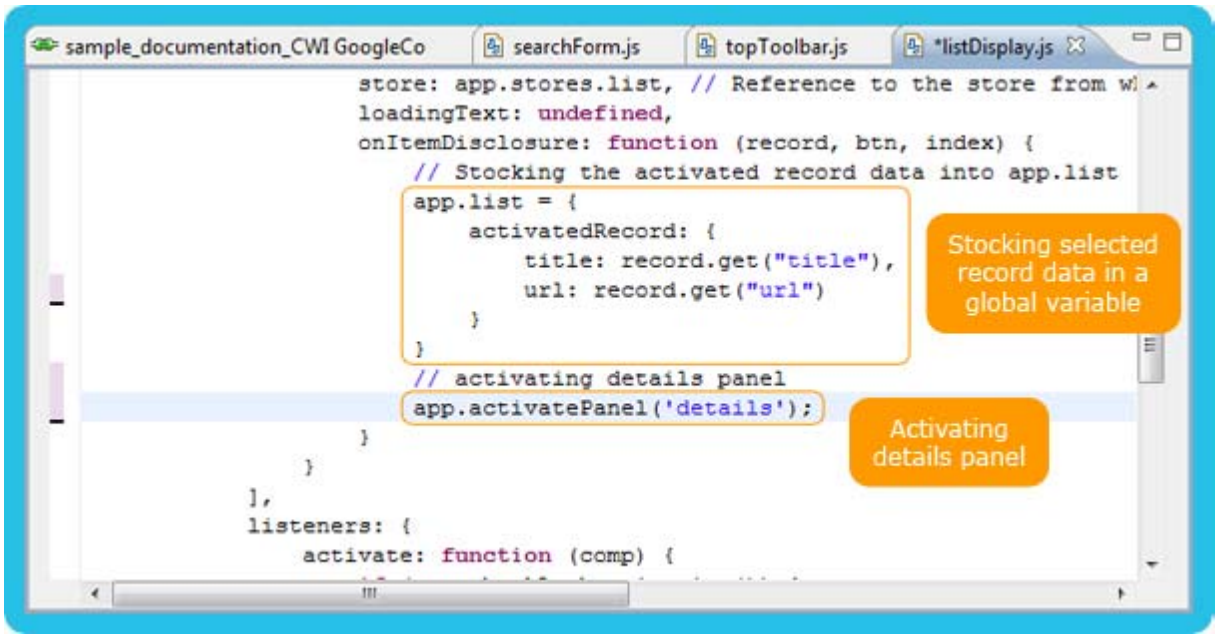


Figure 2 - 80: Updating list item activation behavior in listDisplay.js file

You can find the complete edited `listDisplay.js` file in the example project `sampleDocumentationMobilizerSencha` resources. You can copy and paste source code from it to save time.

This updated source code allows to:

- store selected item's record data, i.e. the two columns returned by the Convertigo transaction, `title` and `url`, in a global variable that will be accessible from the details sheet to be displayed;
- activate the details panel.

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

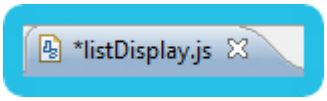


Figure 2 - 81: Edited listDisplay.js file

4 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

Now that the list page is fully updated to display the details sheet on item's activation, we can test again the application to check these last updates.

To execute the application using the test platform

1 If your web browser is closed, launch a research in the mobile application by following steps 1 to 4 of the procedure *"To execute the application using the test platform"* on page 2 - 50.

If your Chrome browser is still opened, simply refresh the test platform page by pressing `Ctrl+R` keys. Then, launch a research in the mobile application by following steps 2 to 4 of the procedure *"To execute the application using the test platform"* on page 2 - 50.

Once all data are loaded, the **Execution result** panel displays the mobile application with the resulting list:

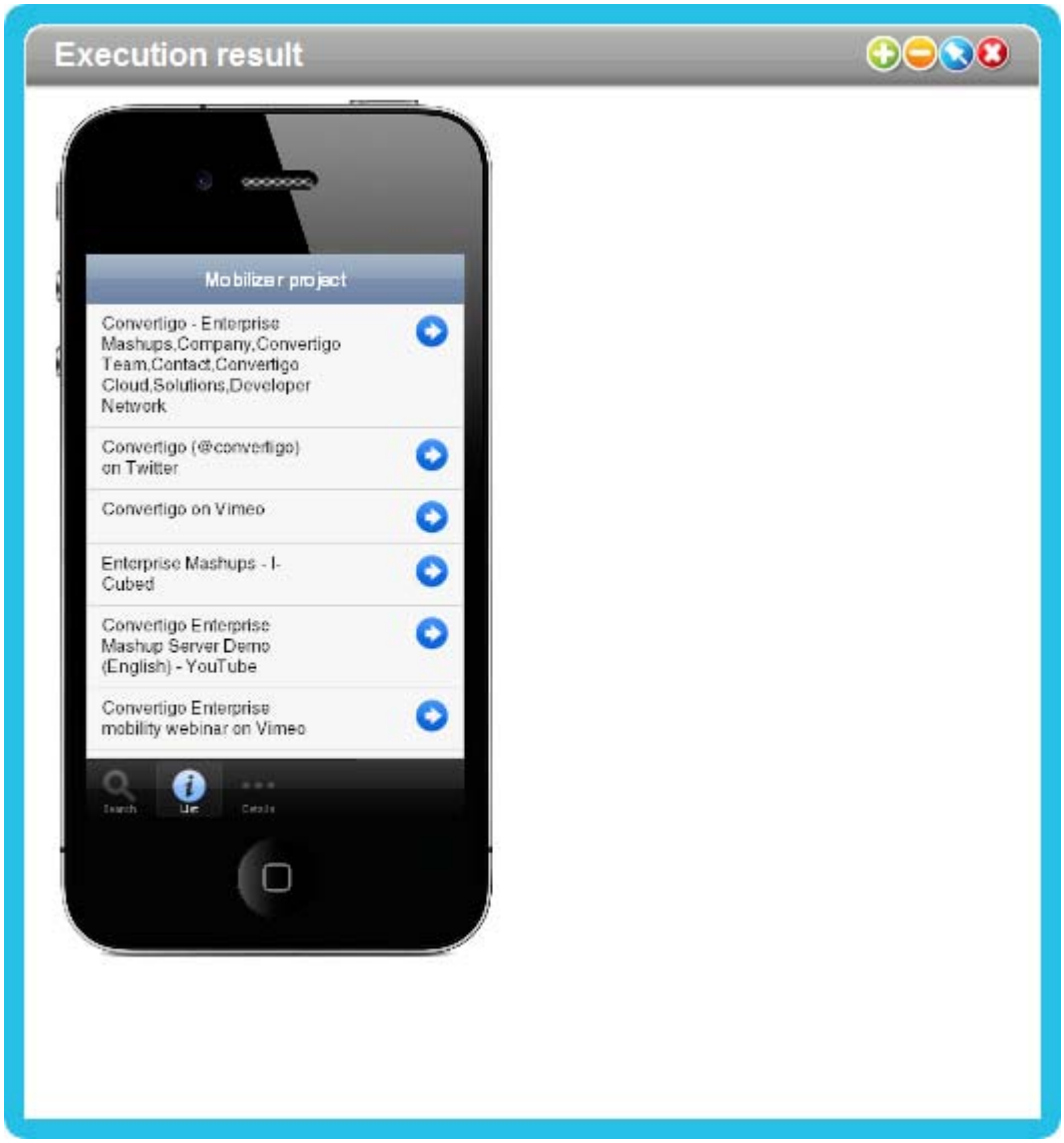


Figure 2 - 82: Result list after search perfomed in iPhone4 device in test platform

2 Click on the right arrow button on one of the items:

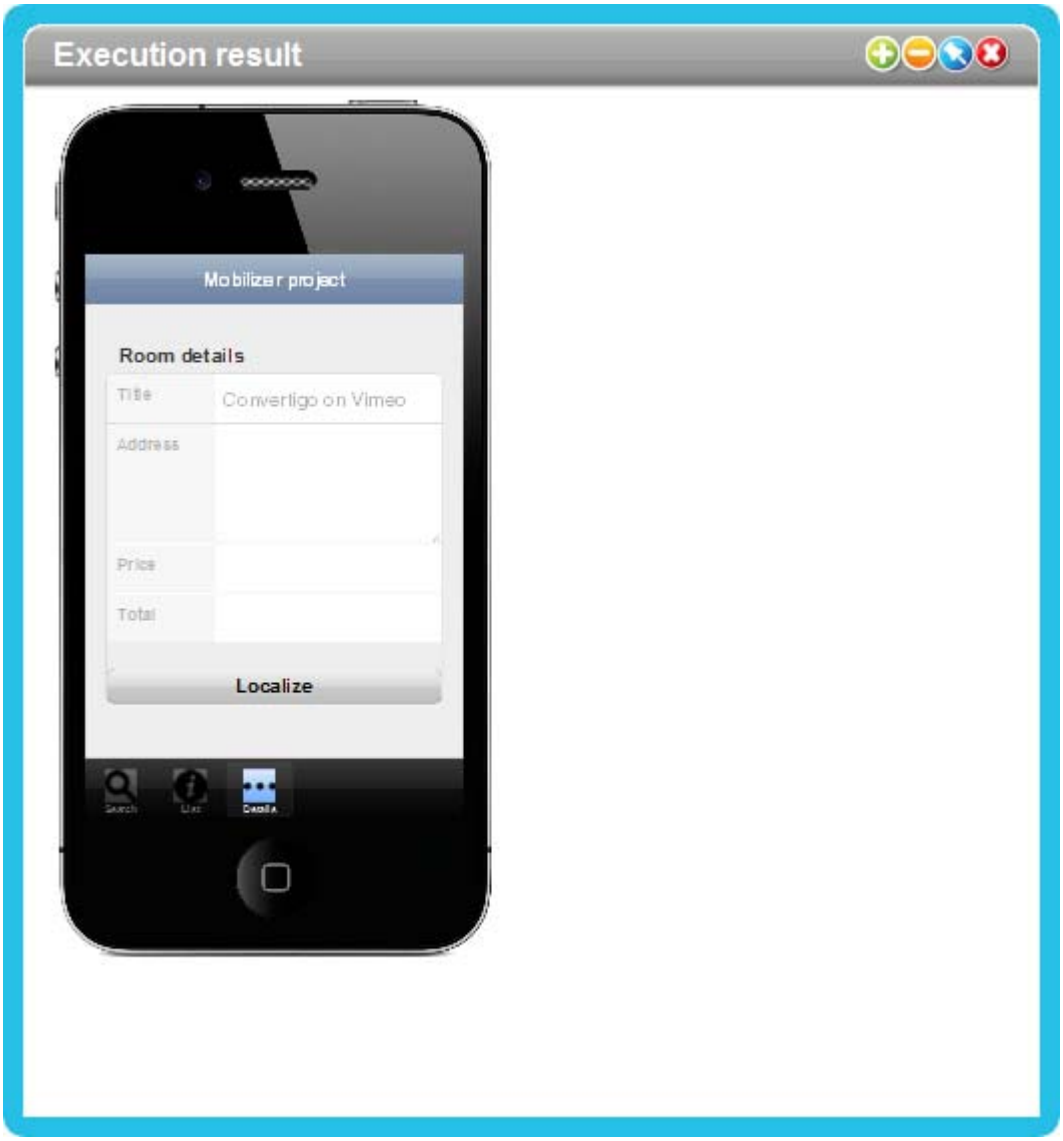


Figure 2 - 83: Details sheet in iPhone4 device in test platform

Details sheet is displayed with default fields corresponding to the template project's data (response of the `LoadList` sequence). The `title` data is already bound because the column's name is the same as in the template project's data. Other fields are empty because columns of data to be displayed are not existing here.

WHAT COMES NEXT?

Now that the list page is fully updated to meat our application's needs, we have to update the details sheet page to display the available data from the list item and to improve these data with the Google Image research, i.e. connecting the details sheet to the second web service providing dynamic data.

SETTING UP THE DETAILS SHEET USER INTERFACE

This section describes step by step how to update the user interface of the details sheet panel to make it match available data: `title` and `URL` columns.

Our application's details sheet should contain:

- a text field for `title` piece of data,
- a description area that will be enhanced later with dynamic data provided by a web service,
- a button to dynamically access the URL contained in `url` piece of data.

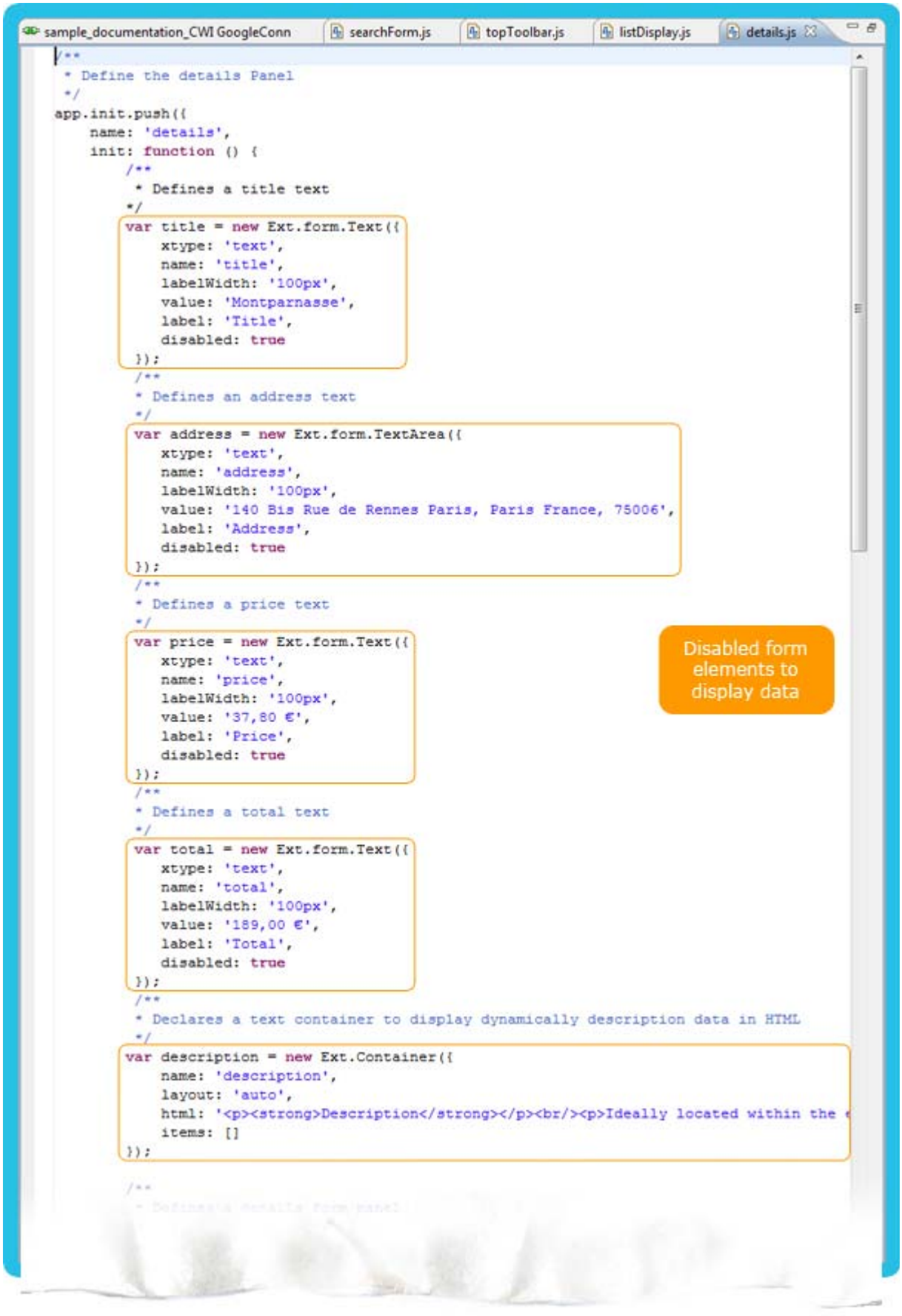
To set up a details sheet page

In previously selected features, the *Details sheet* feature was chosen. A JavaScript file was created in the project environment to implement this page: `details.js`.

1 In the **Project Explorer** view, expand the `sources` folder and double-click on the `details.js` file to open it in the editor.

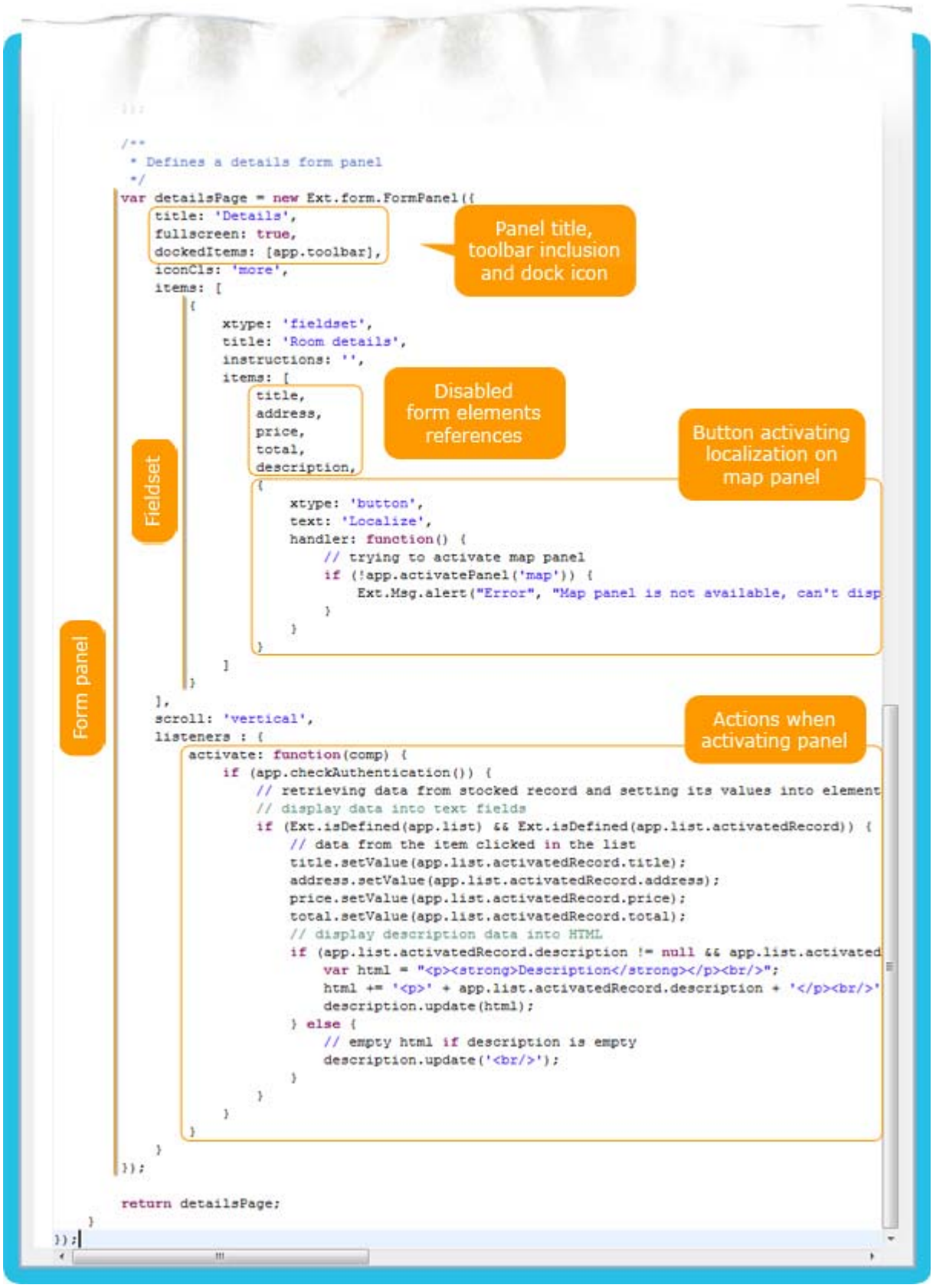
Default source code found in this file corresponds to the basic Mobilizer template application. It is an example of a room booking details sheet containing:

- four disabled form elements and an HTML container declared in order to display data,
- a form panel with:
  - a title
  - the inclusion of the top toolbar,
  - a dock icon,
  - a fieldset including the previously declared fields plus a button allowing to show the localization on the map,
- and the code to execute when activating the panel, which is the code that sets stored list item data into the display elements.




```
/**
 * Define the details Panel
 */
app.init.push({
    name: 'details',
    init: function () {
        /**
         * Defines a title text
         */
        var title = new Ext.form.Text({
            xtype: 'text',
            name: 'title',
            labelWidth: '100px',
            value: 'Montparnasse',
            label: 'Title',
            disabled: true
        });
        /**
         * Defines an address text
         */
        var address = new Ext.form.TextArea({
            xtype: 'text',
            name: 'address',
            labelWidth: '100px',
            value: '140 Bis Rue de Rennes Paris, Paris France, 75006',
            label: 'Address',
            disabled: true
        });
        /**
         * Defines a price text
         */
        var price = new Ext.form.Text({
            xtype: 'text',
            name: 'price',
            labelWidth: '100px',
            value: '37,80 €',
            label: 'Price',
            disabled: true
        });
        /**
         * Defines a total text
         */
        var total = new Ext.form.Text({
            xtype: 'text',
            name: 'total',
            labelWidth: '100px',
            value: '189,00 €',
            label: 'Total',
            disabled: true
        });
        /**
         * Declares a text container to display dynamically description data in HTML
         */
        var description = new Ext.Container({
            name: 'description',
            layout: 'auto',
            html: '<p><strong>Description</strong></p><br/><p>Ideally located within the
            items: []
        });
```

Figure 2 - 84: Default details.js file displayed in editor - part 1



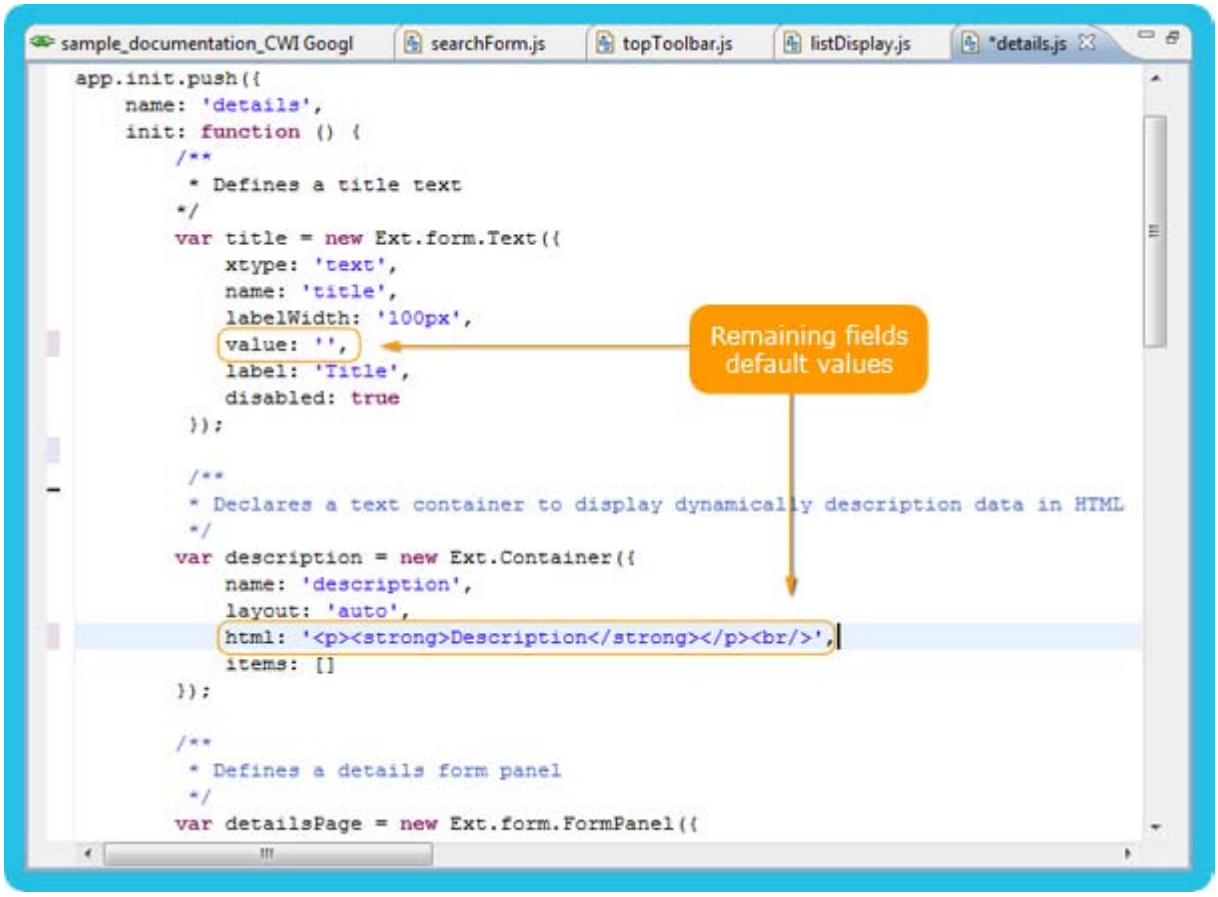

```
/**
 * Defines a details form panel
 */
var detailsPage = new Ext.form.FormPanel({
    title: 'Details',
    fullscreen: true,
    dockedItems: [app.toolbar],
    iconCls: 'more',
    items: [
        {
            xtype: 'fieldset',
            title: 'Room details',
            instructions: '',
            items: [
                title,
                address,
                price,
                total,
                description,
                {
                    xtype: 'button',
                    text: 'Localize',
                    handler: function() {
                        // trying to activate map panel
                        if (!app.activatePanel('map')) {
                            Ext.Msg.alert("Error", "Map panel is not available, can't disp
                        }
                    }
                }
            ]
        }
    ],
    scroll: 'vertical',
    listeners : {
        activate: function(comp) {
            if (app.checkAuthentication()) {
                // retrieving data from stocked record and setting its values into element
                // display data into text fields
                if (Ext.isDefined(app.list) && Ext.isDefined(app.list.activatedRecord)) {
                    // data from the item clicked in the list
                    title.setValue(app.list.activatedRecord.title);
                    address.setValue(app.list.activatedRecord.address);
                    price.setValue(app.list.activatedRecord.price);
                    total.setValue(app.list.activatedRecord.total);
                    // display description data into HTML
                    if (app.list.activatedRecord.description != null && app.list.activated
                        var html = "<p><strong>Description</strong></p><br/>";
                        html += '<p>' + app.list.activatedRecord.description + '</p><br/>'
                        description.update(html);
                    } else {
                        // empty html if description is empty
                        description.update('<br/>');
                    }
                }
            }
        }
    }
});

return detailsPage;
}
});
```

Figure 2 - 85: Default details.js file displayed in editor - part 2

The default source code has to be modified according to the needs of the currently developed application.

In a first time, we have to modify the elements declared at the begining of the file to be able to display the pieces of data our application has to display.

2 We keep the title text field as well as the HTML container, remove the three other fields from the code and edit the two remaining fields default values:




```
app.init.push({
    name: 'details',
    init: function () {
        /**
         * Defines a title text
        */
        var title = new Ext.form.Text({
            xtype: 'text',
            name: 'title',
            labelWidth: '100px',
            value: '',
            label: 'Title',
            disabled: true
        });

        /**
        * Declares a text container to display dynamically description data in HTML
        */
        var description = new Ext.Container({
            name: 'description',
            layout: 'auto',
            html: '<p><strong>Description</strong></p><br/>',
            items: []
        });

        /**
         * Defines a details form panel
         */
        var detailsPage = new Ext.form.FormPanel({
```

Figure 2 - 86: Updating details.js file in editor - part 1

You can find the complete edited `details.js` file in the example project `sampleDocumentationMobilizerSencha` resources. You can copy and paste source code from it to save time.

Beware! Follow the steps of the tutorial, do not copy and paste the complete source code. At this stage, the `panel` object with all its properties and functions is not yet updated.

Then, we add a button that allows the user to access an URL when clicking on it.

3 Modify the code up to get the following source code:

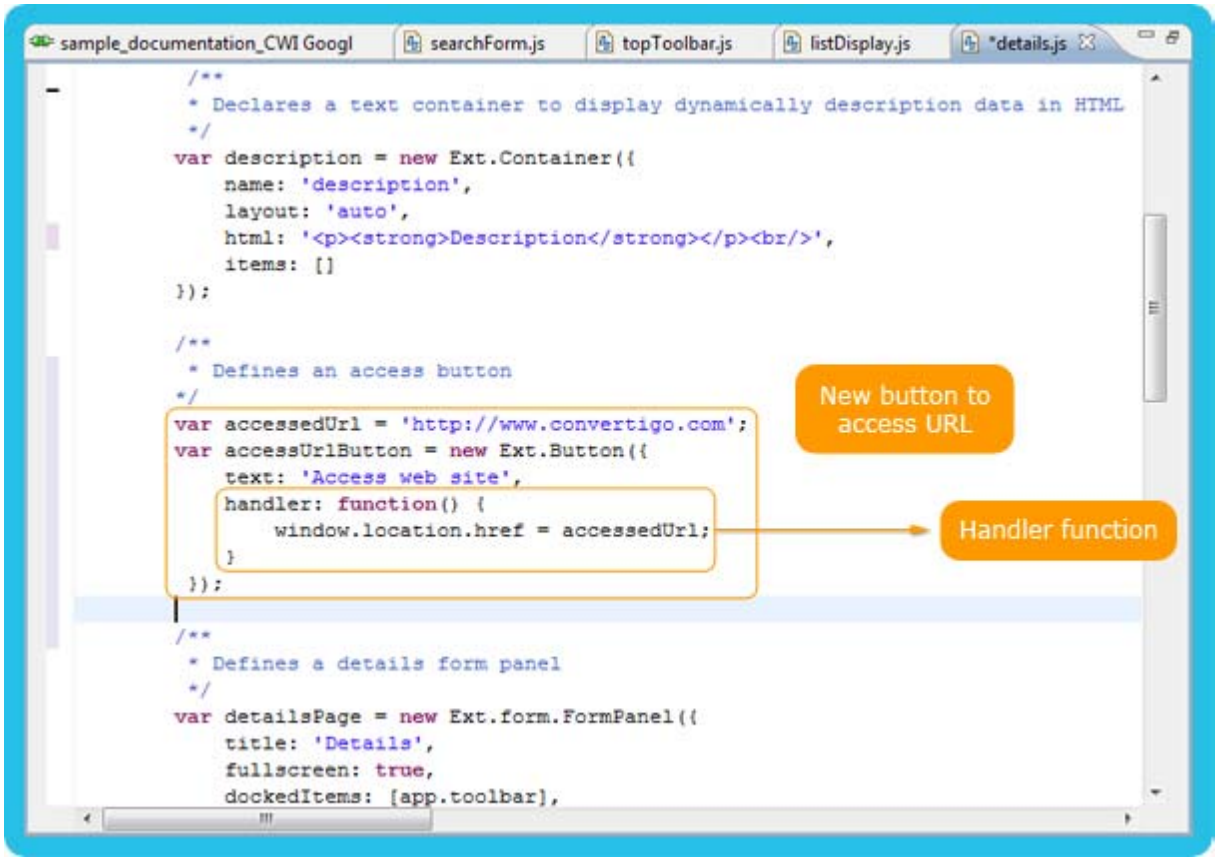


Figure 2 - 87: Updating details.js file in editor - part 2

This updated source code allows to define an URL String object as well as a **new button** object, that accesses to the URL set in the String variable when clicking on it thanks to a handler function.

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

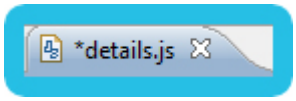


Figure 2 - 88: Edited details.js file

4 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

In a second time, we have to update the fieldset for it to include only the remaining and updated elements, and its display title.

5 Modify the code up to get the following source code:

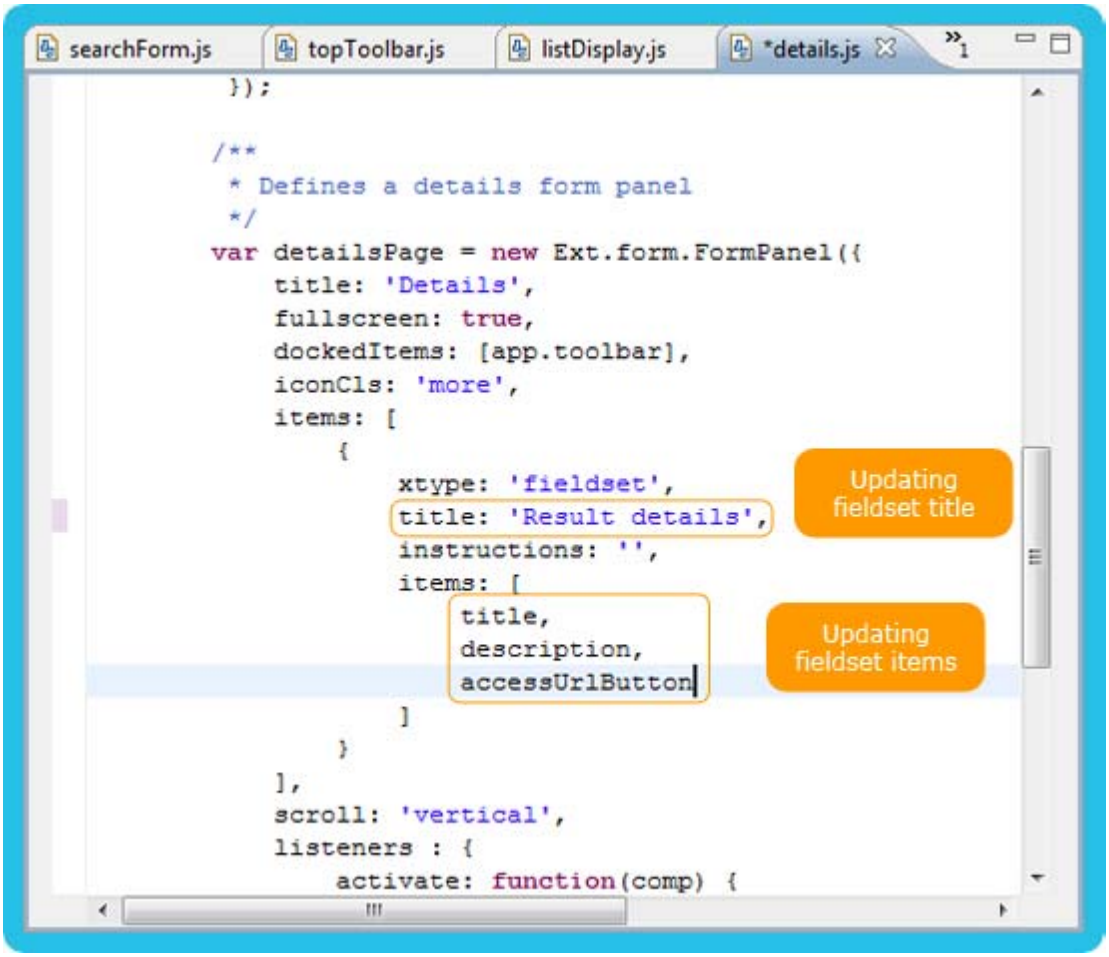


Figure 2 - 89: Updating details.js file in editor - part 3

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

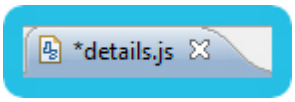


Figure 2 - 90: Edited details.js file

6 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

The final part of the source code is the part that, on the activation of the panel (`activate` event), sets the data stored from the list's selected item into the elements displayed in the panel. The template existing code already sets data but has to be updated to match the newly updated elements.

7 Modify the code up to replace the `if(app.checkAuthentication())` inner code by the following source code:

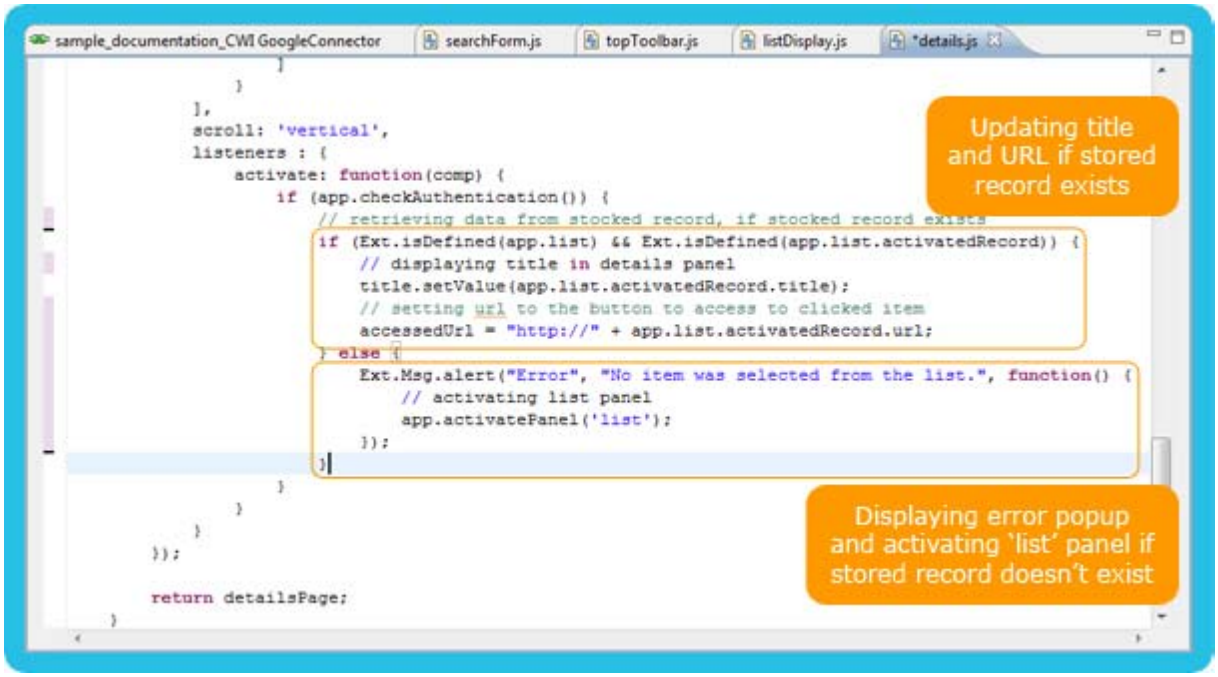


Figure 2 - 91: Updating details.js file in editor - part 4

*The `app.checkAuthentication()` method is testing that the user is well authentified in the login panel before anything else, in case the **Login form** feature has been selected in the project creation wizard.*
*In our application, the **Login form** feature is not implemented, the `app.checkAuthentication()` method will therefore always return `true`. We can keep it in source code, it has no effect.*

The modified piece of source code tests if a record is stored in the proper global variable and:

- if a record is stored:
  - updates the title text field value with the stored one;
  - updates the url String variable used by the button click handler function with the stored url.
- if no record is stored:
  - displays an error message;
  - re-activates list panel when message popup is validated.

Beware! Follow the steps of the tutorial, do not copy and paste the complete source code. At this stage, the web service call adding dynamic details data is not yet implemented.

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

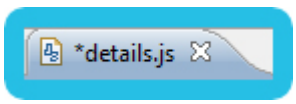


Figure 2 - 92: Edited details.js file

8 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

Now that the details sheet page is updated to display the selected item's data, we can test again the application to check these last updates.

To execute the application using the test platform

1 Follow steps 1 to 2 of the procedure *"To execute the application using the test platform"* on page 2 - 73.

Figure 2 - 93: Details sheet in Android device in test platform

Details sheet is displayed with updated elements:

- updated title,
- empty description HTML container (for further use),
- button that allows to access dynamically to the result URL.

2 Click on the **Access web site** button:

Figure 2 - 94: Accessing target web site in test platform

The target web site corresponding to the selected item from the list is opened in the mobile device.

You can notice that the test platform allows accessing the web site but it is not built to navigate then easily in the web site. It will be more useful in a real mobile device, see procedure .

WHAT COMES NEXT?

Now that the details page is updated to display the result item details and to access target web site, we have to update the details sheet page to improve these data with the Google Image research, i.e. connecting the details sheet to the second Convertigo web service providing dynamic data.

INTEGRATING A DETAILS SHEET WITH A CALL TO CONVERTIGO AND ITS RESPONSE

This section describes how to integrate the response of a call to a Convertigo transaction (or sequence) with a details sheet panel.

To integrate a call to a Convertigo transaction and its response in details sheet

This procedure completes *"To set up a details sheet page"* on page 2 - 76 as it will finish editing the details sheet feature.

1 Switch back to Convertigo Studio where `details.js` file was opened for edition.

The `activate` event handler function was last edited to display the data stored from the list's selected item into the panel elements, if a stored record was existing. In this case, we want to add a call to a Convertigo transaction and retrieve and display its response in the application.

2 Modify the code up to add the following source code in the `if` case:

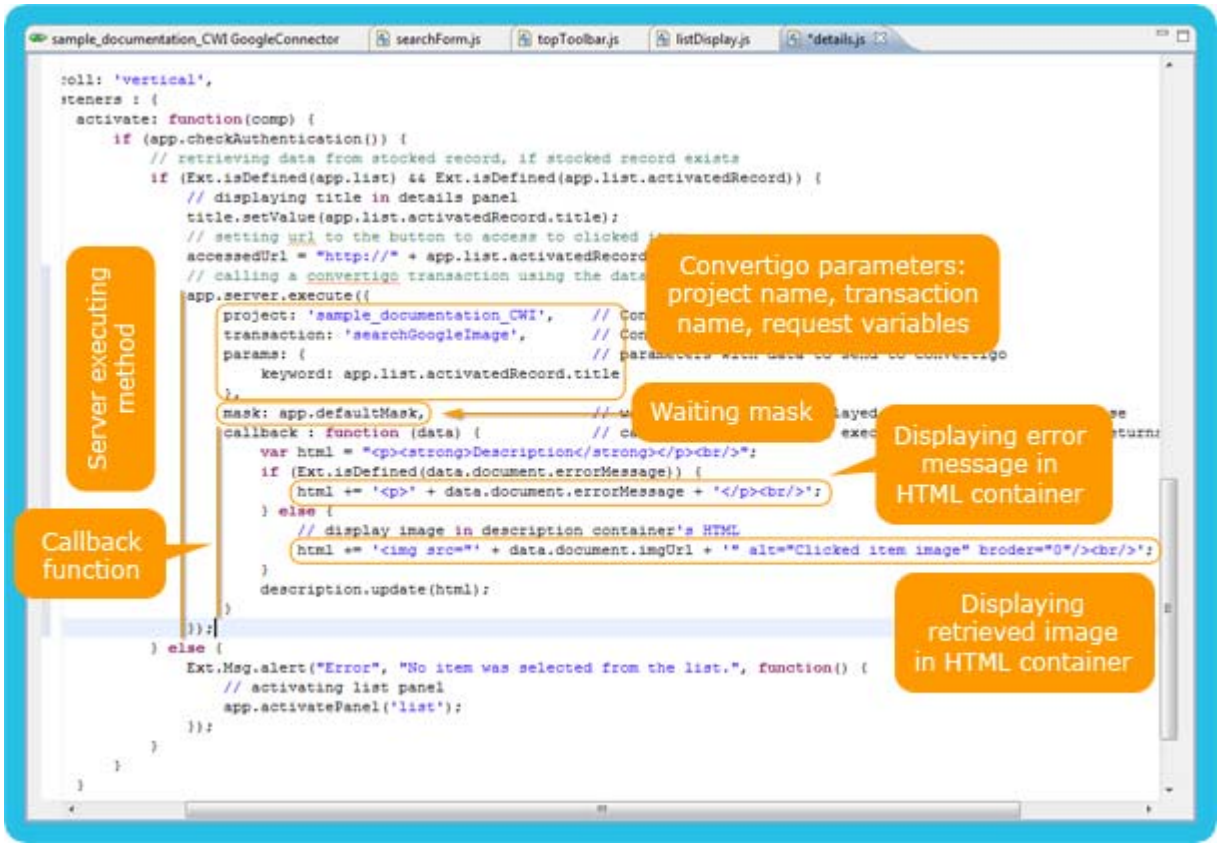


Figure 2 - 95: Updating details.js file in editor - part 5

*`C80 Server` object, offered by **Convertigo mobilelib** library, is derived from **Sencha Touch** `Observable` object. It represents a Convertigo Server and allows calling a transaction or a sequence and retrieving its result after its execution thanks to the `execute` method.*
For detailed information on the `Observable` object, with all its properties and methods, refer to the Sencha Touch API documentation at: http://dev.sencha.com/deploy/touch/docs/?class=Ext.util.Observable

The added piece of source code is the server executing method. This method is separated in three parts:

- the Convertigo parameters definition:
  - allows to target a project different than the current Mobilizer project (in our example, the CWI `sample_documentation_CWI` project);

- then to define the name of the transaction (or sequence) to request (in our example, the `searchGoogleImage` transaction), as well as the connector name, if it is not the default connector (in our case it is the default connector of the project, so it is not precised here);
- and the variables to send with the request to Convertigo and their values (here by declaring a `params` object containging the one requested parameter `keyword` and its value, the `title` from stored record).

- the waiting `mask`: allows to define a mask to be displayed while the transaction or sequence is executed on Convertigo Server and until the response is retrieved. Here, we only bind this property with a mask defined in the `app` object. For more information about the `app.js` file, see *"SOURCES directory"* on page 2-28.
- the `callback` handler function: allows to define the actions to perform when the response is back from Convertigo.
  - Here, we simply update the HTML container content with the error message in case of error or with the image retrieved from the response if no error is found.

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

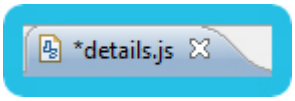


Figure 2 - 96: Edited details.js file

3 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

Now that the details sheet page is updated to call a transaction and display its response in addition to previously displayed elements, we can test again the application to check these last updates.

To execute the application using the test platform

1 Follow steps 1 to 2 of the procedure *"To execute the application using the test platform"* on page 2 - 73.

After clicking on the right arrow button of one item of the list, the details sheet is activated in the mobile application displayed in the **Execution result** panel. The loading mask is automatically displayed while the transaction is executed on Convertigo:

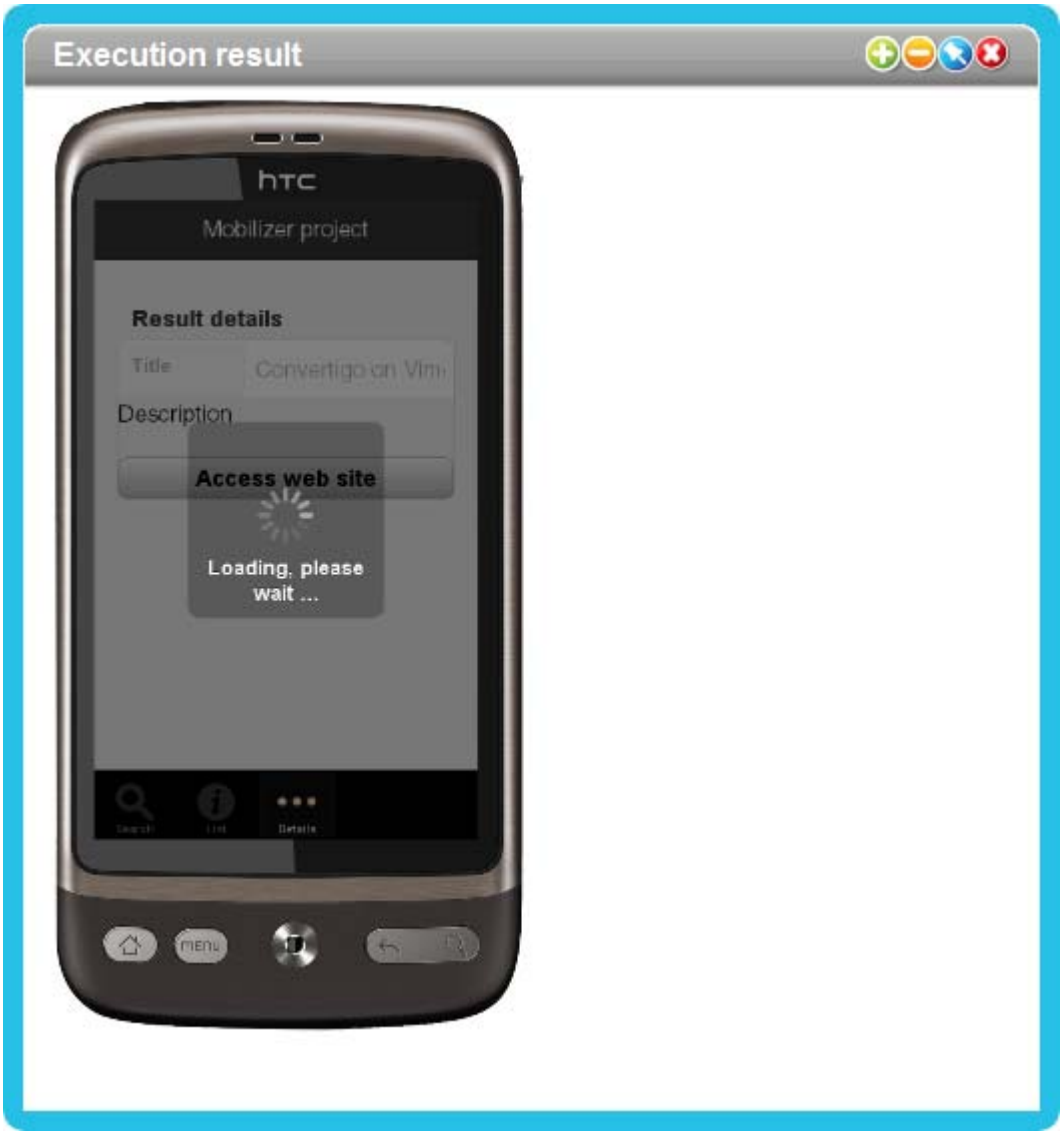


Figure 2 - 97: Activation of loading mask on details sheet

The mask activation allows preventing the user to perform any action on the application while loading data from Convertigo transaction execution. The mask activation and deactivation are automatically synchronized with the `server.execute` method and its loading process.

Once data are retrieved from Convertigo, the **Execution result** panel displays the mobile application with the resulting image added to existing data:

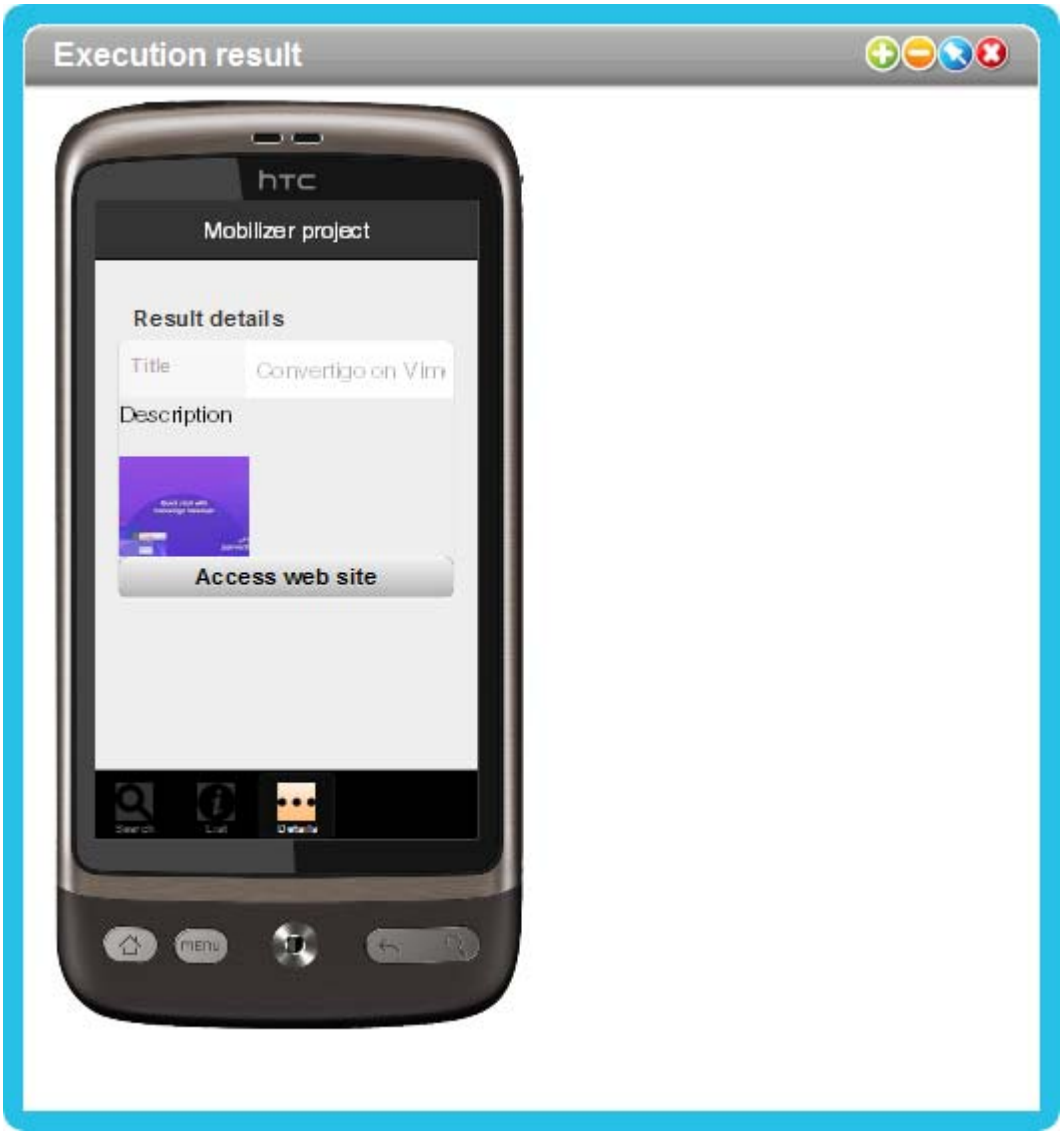


Figure 2 - 98: Details sheet after image search perfomed in test platform

WHAT COMES NEXT?

The implementation of our application is now finished and the application is fully functional. Let's approach the application access directly on smartphones.

2.4 Use the application from your own mobile device

As previously explained, there are two ways of running the developed mobile application on your own mobile device: running the *web application* or installing the *native application*. For more information about this possibility, see *"Convertigo Mobilizer architecture"* on page 1-2.

Before running the application from a smartphone, it has to be accessible from anywhere, so the projects have to be deployed on a Convertigo Cloud server.

This section will present both ways of running and using the developed mobile application, as well as how to deploy projects in a Convertigo Cloud server:

- Deploying projects in a Convertigo Cloud server

- Running mobile web application
- Building and installing native mobile application

2.4.1 Deploying projects in a Convertigo Cloud server

As the `sampleMobilizerSencha` Mobilizer project uses the `sample_documentation_CWI` Web integrator project, both projects have to be deployed on the Convertigo server for the mobile application to keep working.

To deploy a project from a Convertigo Studio to a Convertigo Server

1 In the **Projects** view, right-click on the project to be deployed (for example `sample_documentation_CWI` project) and select **Deploy**.

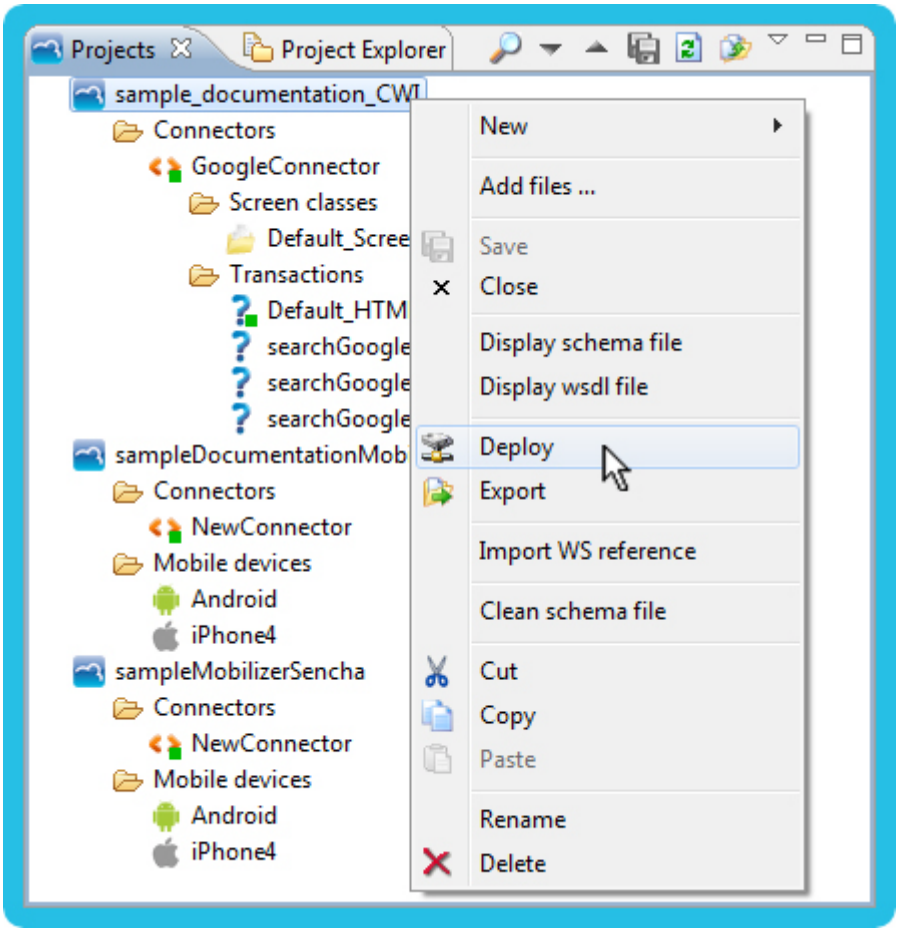


Figure 2 - 99: Deploying a Convertigo project

The **Deploy a Convertigo project** window opens, depending on the type of user:

Case of a Convertigo Studio testing user

While downloading and installing the Convertigo Studio, you registered and, thanks to that, you were granted a free access to trial Convertigo Shared Cloud. You received an email of this form, containing a registration certificate:

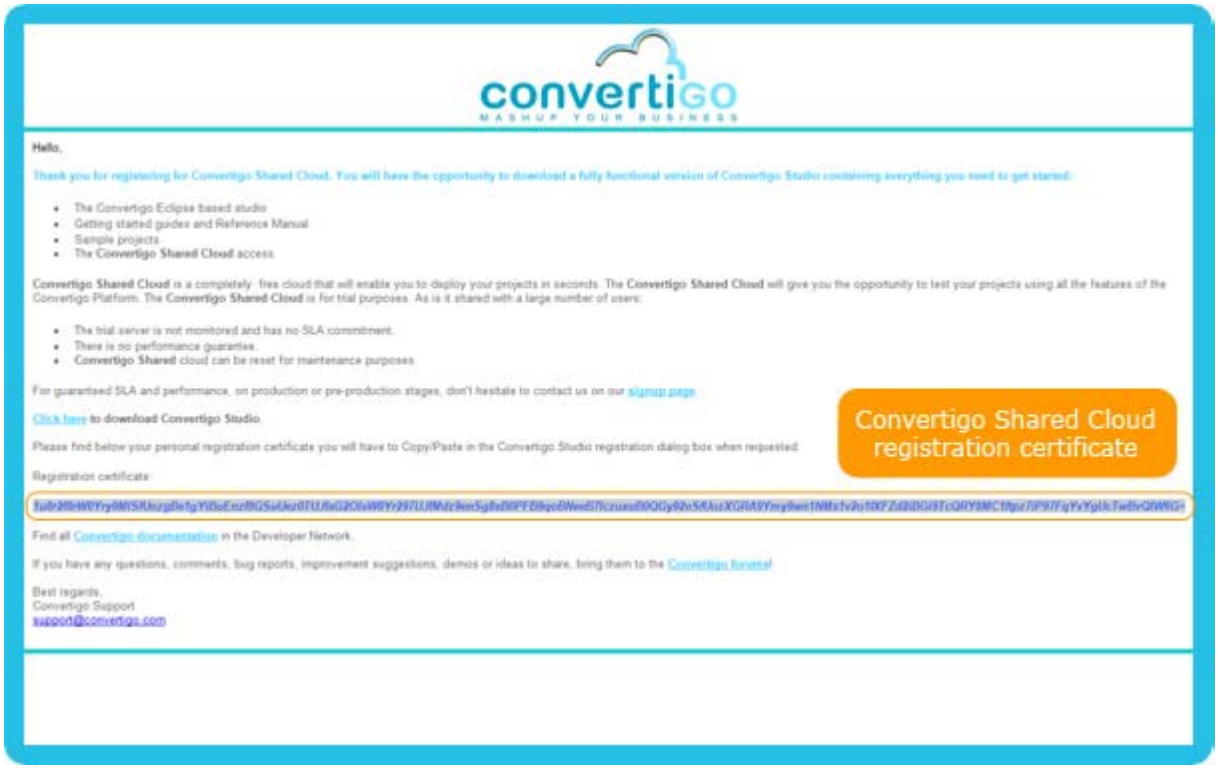


Figure 2 - 100: Email with Convertigo Shared Cloud registration certificate

When starting the Convertigo Studio for the first time, you entered the registration certificate in the **Trial registration certificate** window:

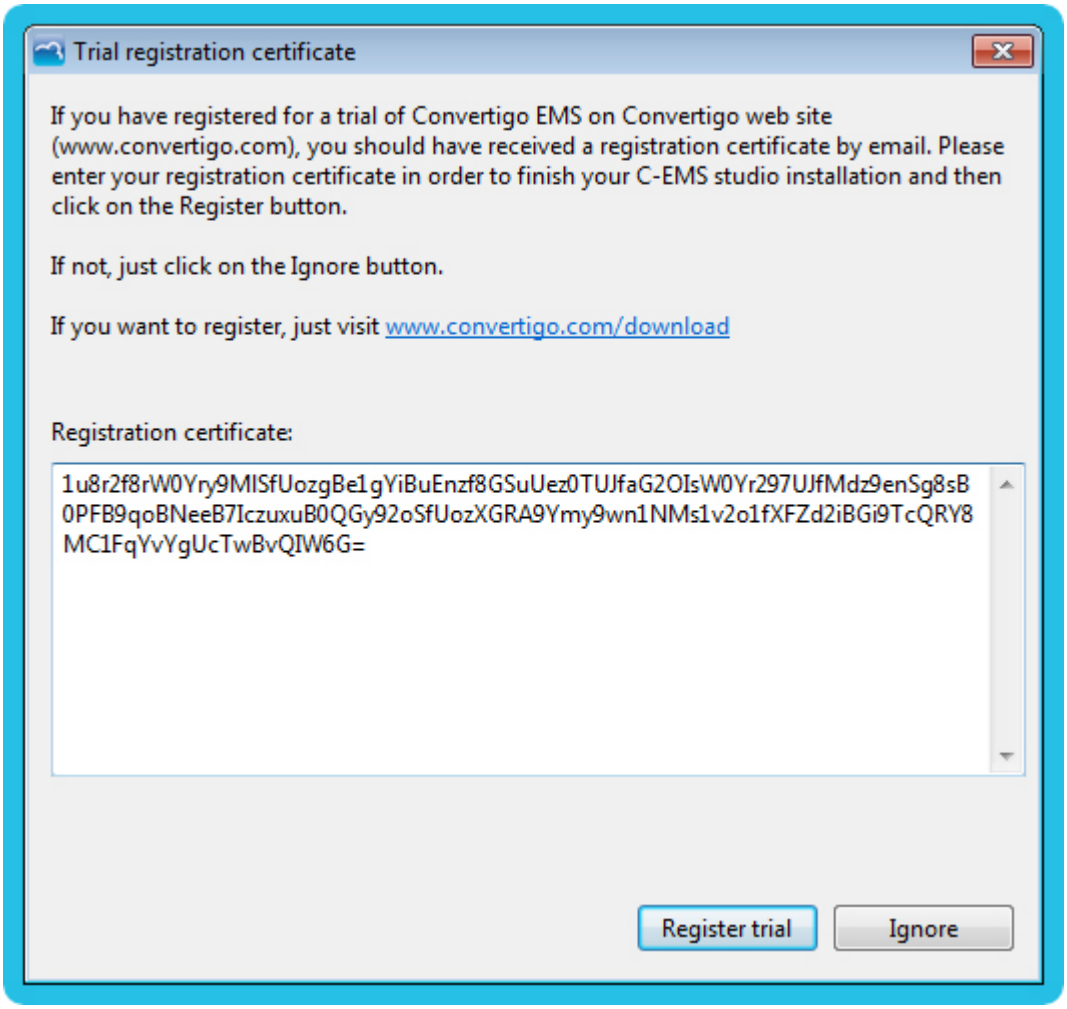


Figure 2 - 101: Registrating Convertigo Shared Cloud in Studio

For this type of user, the **Deploy a Convertigo project** window opens with target Server data fields automatically filled with the Convertigo Shared Cloud server, the user's email address as username and the generated password:

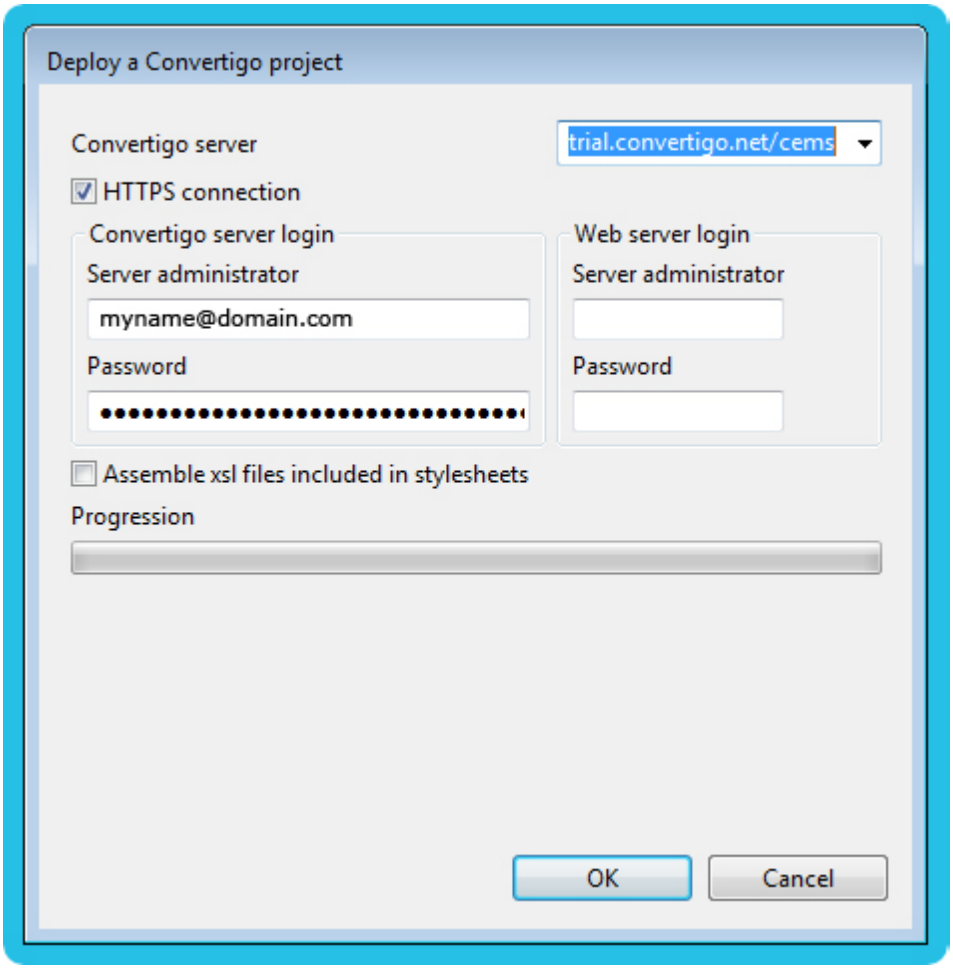


Figure 2 - 102: Deploy window automatically filled

Case of a Convertigo Cloud client

If you ordered for a Convertigo Cloud server, you have an email with the URL and username / password of your Cloud server administration.

For this type of user, the **Deploy a Convertigo project** window opens empty:

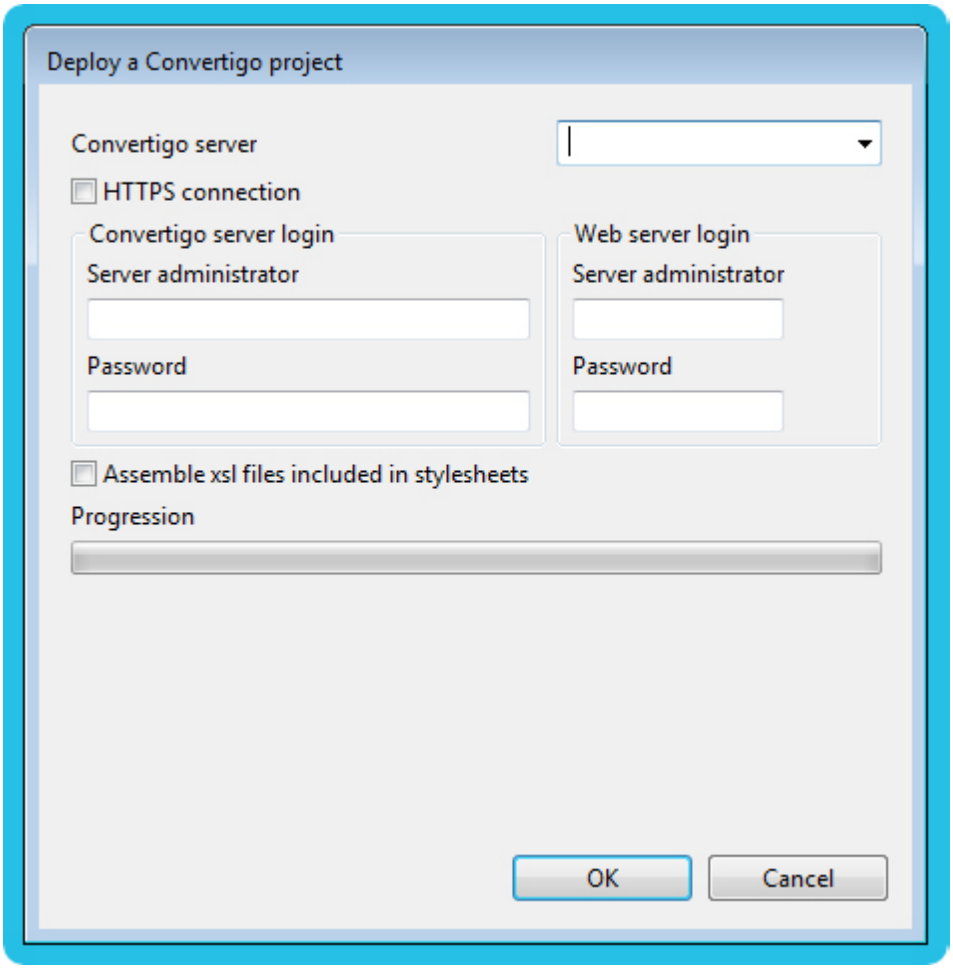


Figure 2 - 103: Empty Deploy window

2 Enter your Convertigo Cloud server data in the fields, with:

- the Convertigo server address in the **Convertigo server** field,
- the Convertigo Server login credentials in the **Server administrator** and **Password** fields,
- possibly checking **HTTPS connection** checkbox:

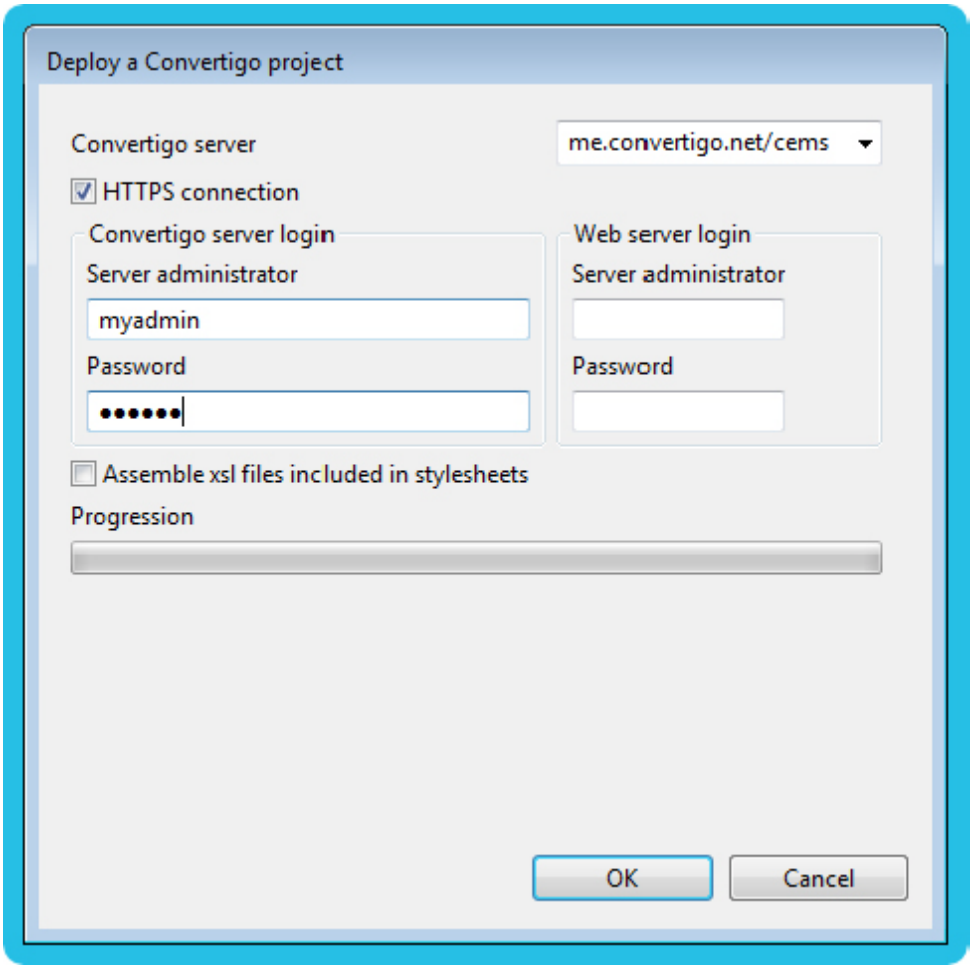


Figure 2 - 104: Filling fields in deploy window

Continuing the procedure for both users:

3 For all users, click on **OK** to validate the deployment.

The **Progression** bar progresses while the project is being deployed on the server:

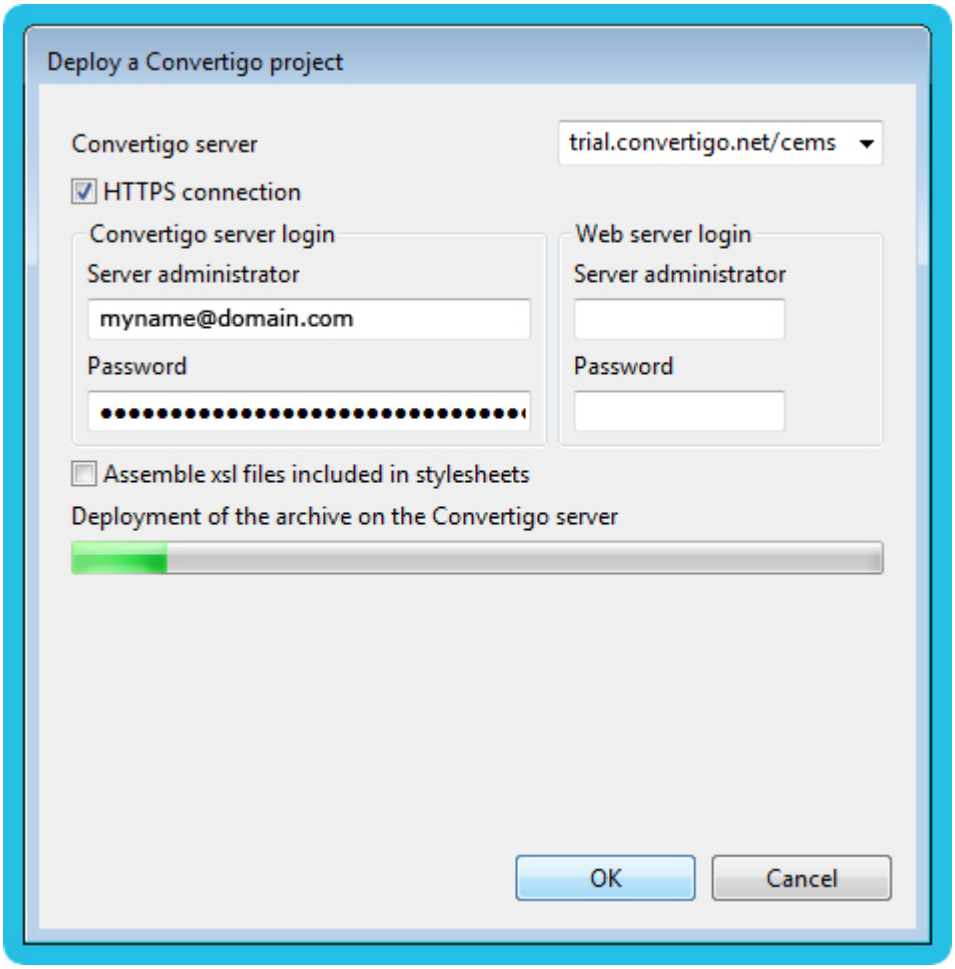


Figure 2 - 105: Progression of project deployment

When the project is sucessfully deployed, a confirmation message is displayed in a popup window:

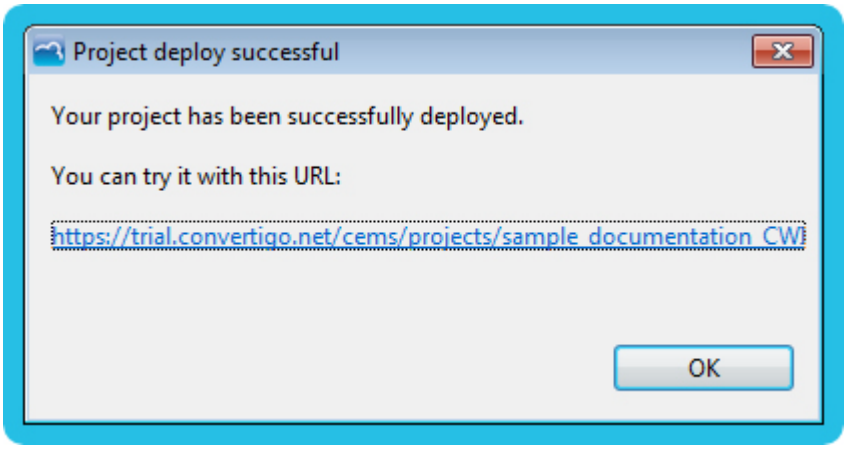


Figure 2 - 106: Project deployment confirmation message

The project is deployed on the Convertigo Cloud server.

4 Click on **OK** to close the popup.

5 Repeat steps 1 to 4 of this procedure for deploying the second project `sampleMobilizerSencha`.

WHAT COMES NEXT?

Now that both projects are deployed on a Convertigo Cloud server, i.e. they are accessible from any place on the Internet, let's see how to access the mobile web application from a smartphone.

2.4.2 Running mobile web application

This section explains step by step how to run and use the web version of the developed mobile application.

To run a web mobile application on a smartphone

1 Launch the test platform of the Mobilizer project deployed on your Convertigo Cloud server following the procedure *"To open the test platform of the project"* on page 2 - 45.

If you are not using a Convertigo Cloud server, use the external hostname or IP address of the computer where Convertigo runs, so that your mobile device can access the host.

For example, use an address of this type:

```
http://trial.convertigo.net/cems/projects/sampleMobilizerSencha
```

2 Expand a mobile device panel (for example `Android`).

3 Use the camera in your smartphone with the appropriate application to flash the QR code displayed on the right part (entitled **Web Application**):

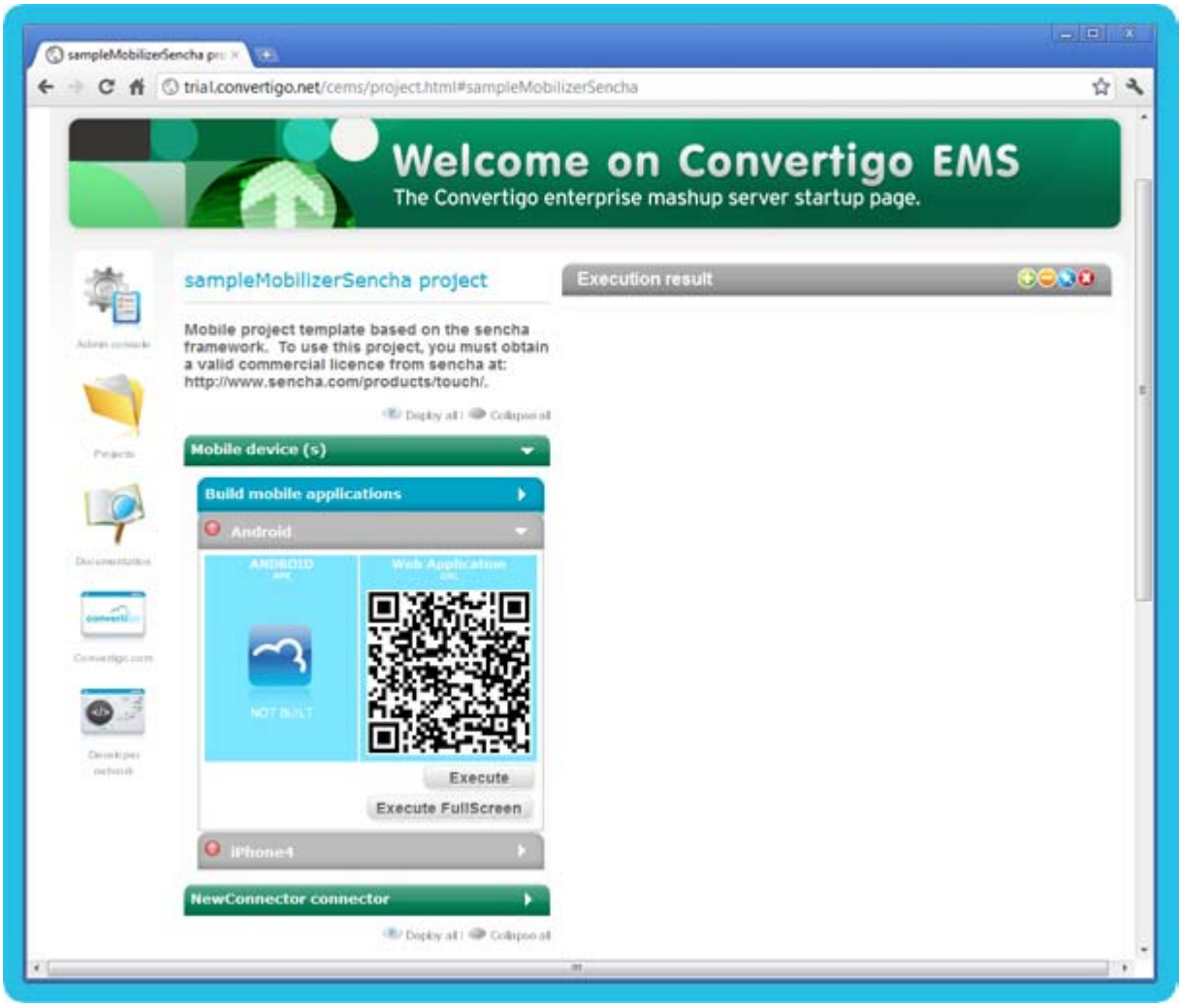


Figure 2 - 107: Flashing mobile web application QR code with device

The web application opens in your device's browser and you are able to use it, provided that the browser of your mobile device is Webkit-based.

You can flash the Web Application QR code of any device present in the test platform with any mobile device: the URL encoded in these QR codes is the same for all devices.
You can use the same Web application even on a mobile device that was not added to the initial project.

WHAT COMES NEXT?

Now that you can access the mobile web application from your smartphone, let's see how to build and install the native mobile application on a smartphone.

2.4.3 Building and installing native mobile application

This section explains step by step how to build the native mobile applications for all devices and to install the application on a smartphone.

Before building the application with Convertigo Mobilizer using ***PhoneGap***, you have to configure your ***PhoneGap*** account in Convertigo through Convertigo Server administration.

To configure PhoneGap account in Convertigo Mobilizer

This feature is available on Convertigo Server only for Convertigo Cloud clients, i.e. users owning a Convertigo Cloud instance, contrary to Convertigo Shared Cloud testers. Indeed, it is not possible to change options on trial Convertigo Cloud administration as this Convertigo Cloud instance is shared by all testing users.

For these testing users, this configuration can be set on the Convertigo Studio admnistration, i.e. the administration of the Convertigo Server embedded in the Convertigo Studio.

1. Launch a standard Web browser (Mozilla Firefox for example).
2. In the URL address field, type in the URL in the following format:

```
http://<ConvertigoServer>/<ConvertigoAppName>/admin
```

For example, to access a local Convertigo Studio administration:

- *ConvertigoServer* = `localhost:18080` (do not forget to position the port in local installation configuration)
- *ConvertigoAppName* = `convertigo`

It gives, in a Mozilla Firefox URL address bar:

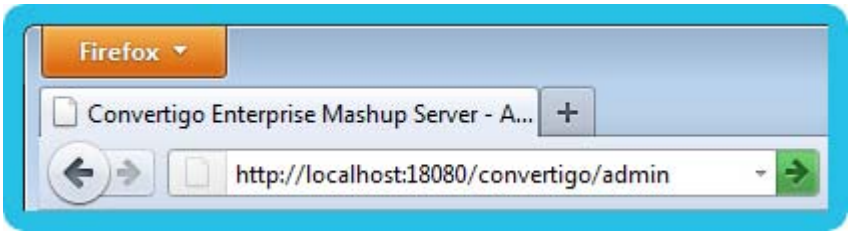


Figure 2 - 108: Convertigo Studio administration URL

For example, to access a Convertigo Cloud administration:

- *ConvertigoServer* = `me.convertigo.net`
- *ConvertigoAppName* = `cems`

It gives, in a Mozilla Firefox URL address bar:

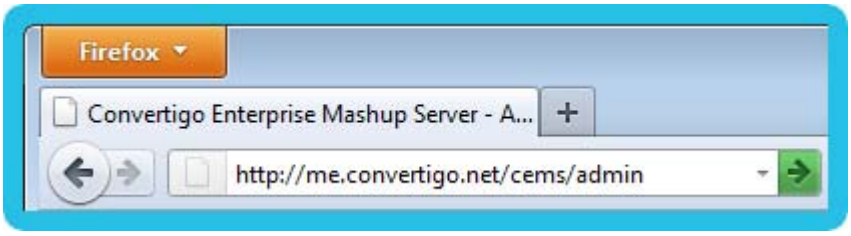


Figure 2 - 109: Convertigo Cloud administration URL

The administration authentication page opens:

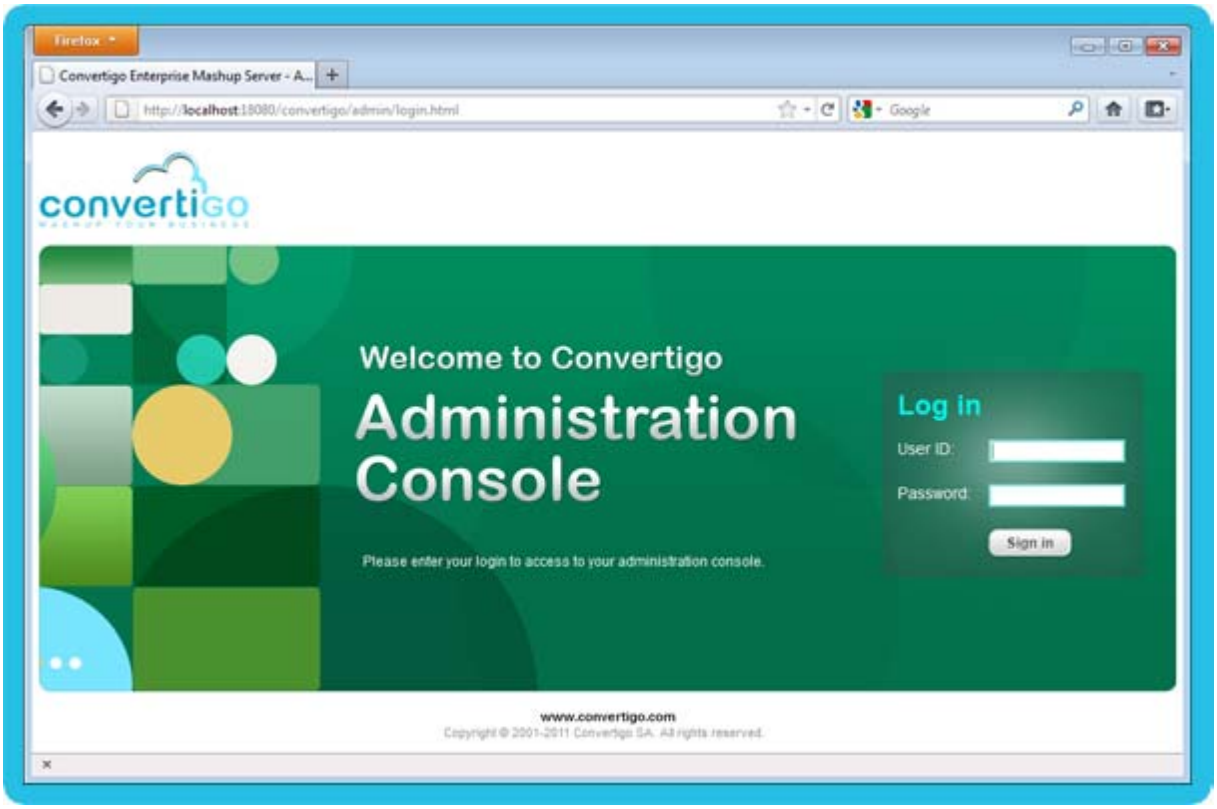


Figure 2 - 110: Convertigo Server administration authentication page

3 Enter the Convertigo Server login credentials in the **User ID** and **Password** fields. For example, in case of Convertigo Studio, default username / password is `admin` / `admin`. In case of a private Convertigo Cloud, credentials were delivered by email.

4 Click on the **Sign in** button or validate by pressing `Enter` key.

The administration Home page opens:

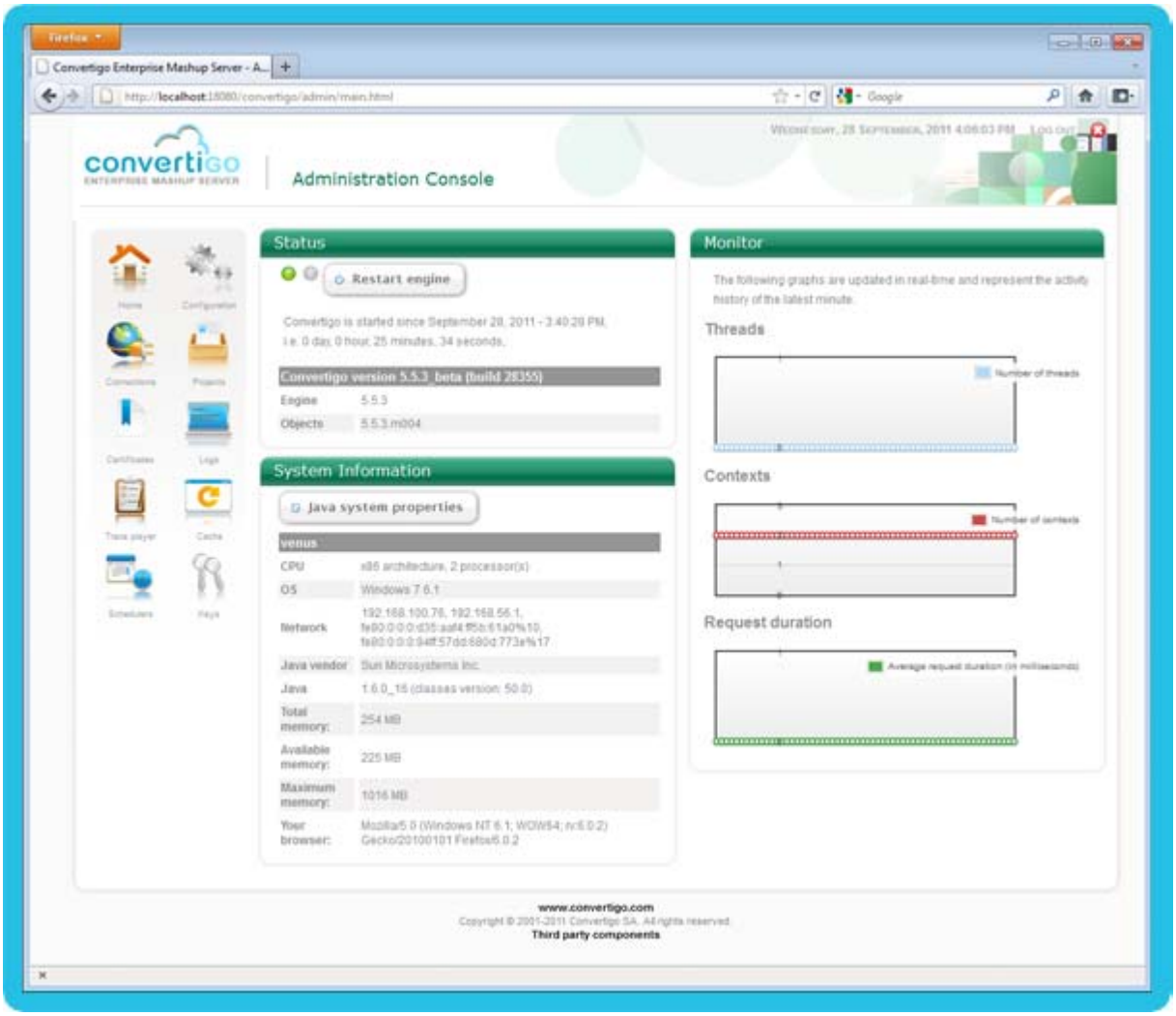


Figure 2 - 111: Convertigo Server administration Home page

5 Click on the **Configuration** button in the left menu to access the Configuration page:

Figure 2 - 112: Accessing Configuration page

6 Click on the Mobile Builder section to expend it:

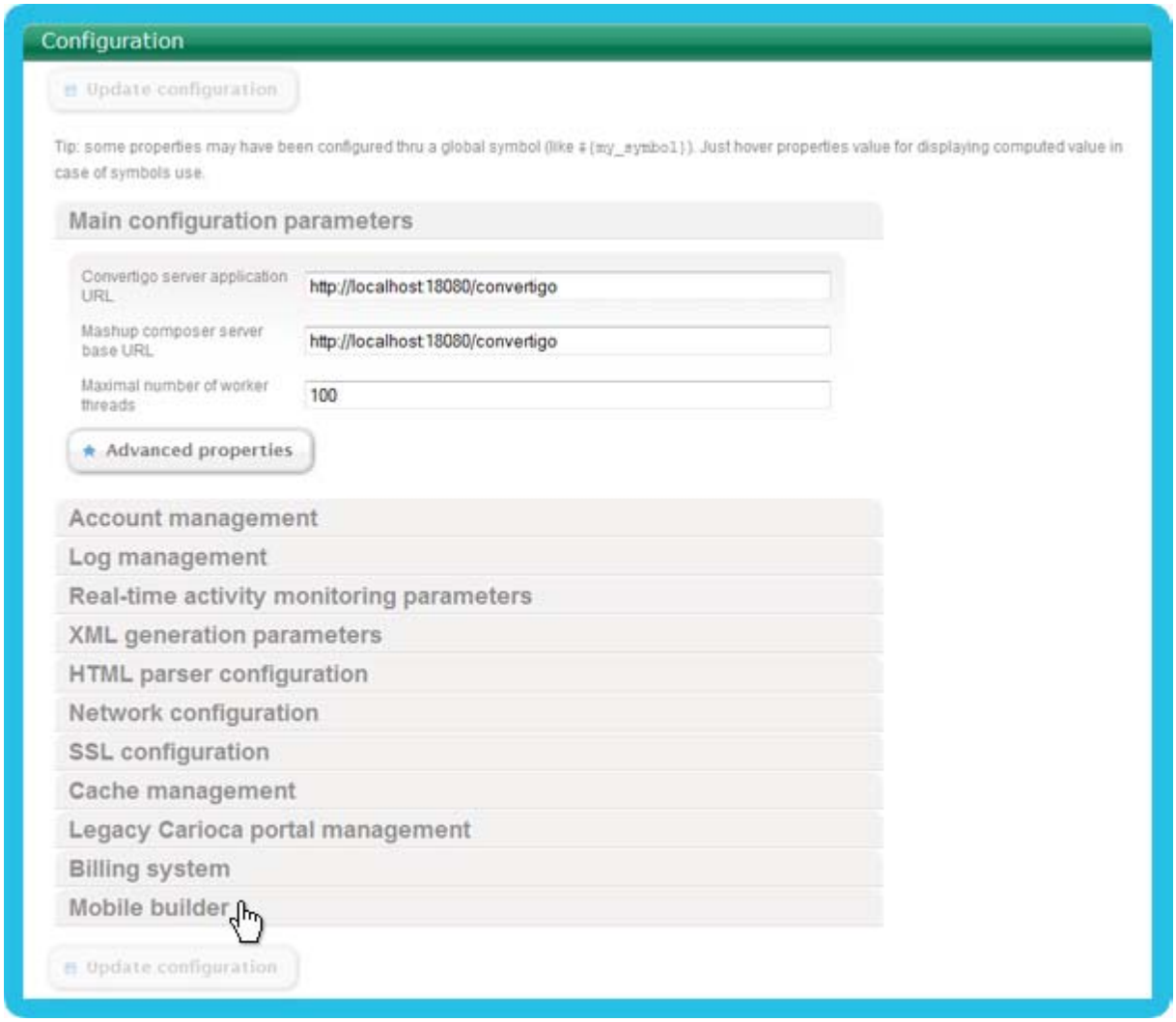


Figure 2 - 113: Expending Mobile Builder section in Configuration page

7 Fill in the **Mobile builder username** with the ***PhoneGap build*** account (email address) and the **Mobile builder password** with ***PhoneGap build*** password:

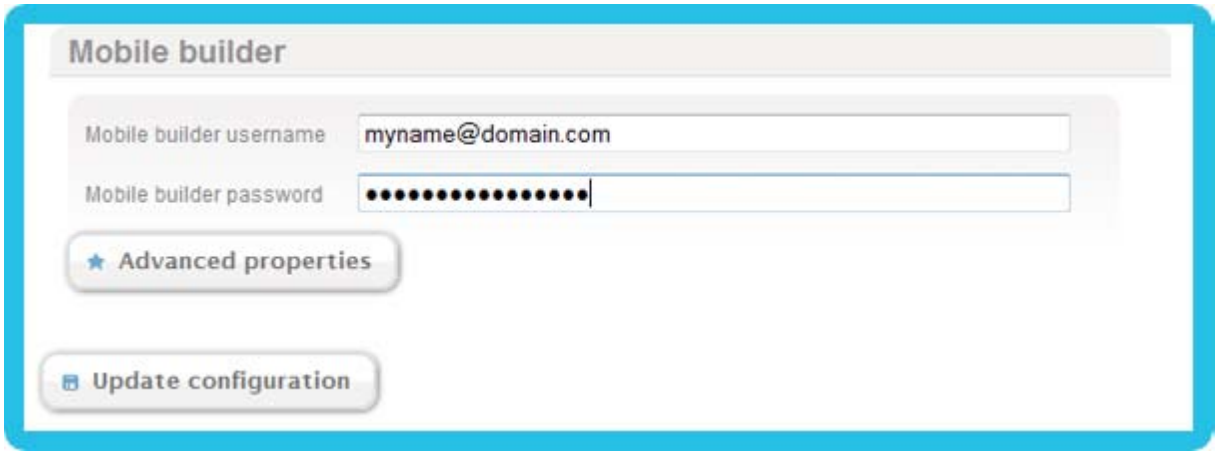


Figure 2 - 114: Updating PhoneGap build account

8 Click on the **Update configuration** button to validate the change of ***PhoneGap build*** account in Convertigo.

9 A validation message is displayed in a popin window:

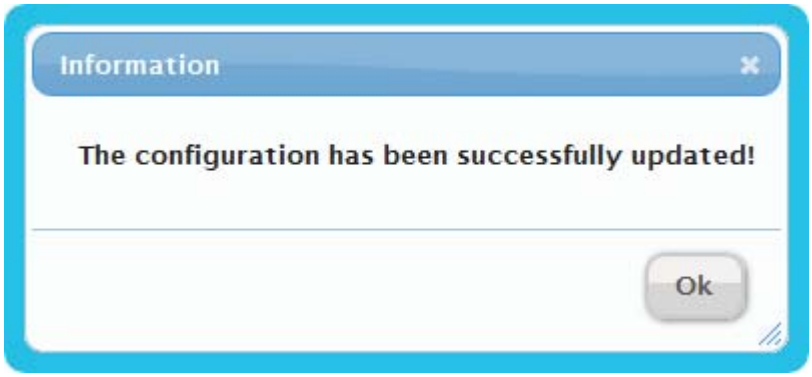


Figure 2 - 115: Updated configuration confirmation message

10 Click on **Ok** to close the window.

The Convertigo Server is updated, you can continue with building the native applications.

To build mobile devices native applications

1 Launch the test platform of the Mobilizer project deployed on the previsouly updated Convertigo Server (with correct ***PhoneGap*** account configuration) following the procedure *"To open the test platform of the project"* on page 2 - 45.

If you are not using a Convertigo Cloud server, use the external hostname or IP address of the computer were Convertigo runs, so as your mobile device can access this host.

For example, use an address of this type:

```
http://me.convertigo.net/cems/projects/sampleMobilizerSencha
```

You can run the test platform of your project opened in the Studio, by using the `localhost:18080` server host and port in the test platform URL. You will then be able to configure later the mobile application access external URL or URL pointing to the Convertigo Server project.

2 Expand a mobile device panel (for example `Android`):

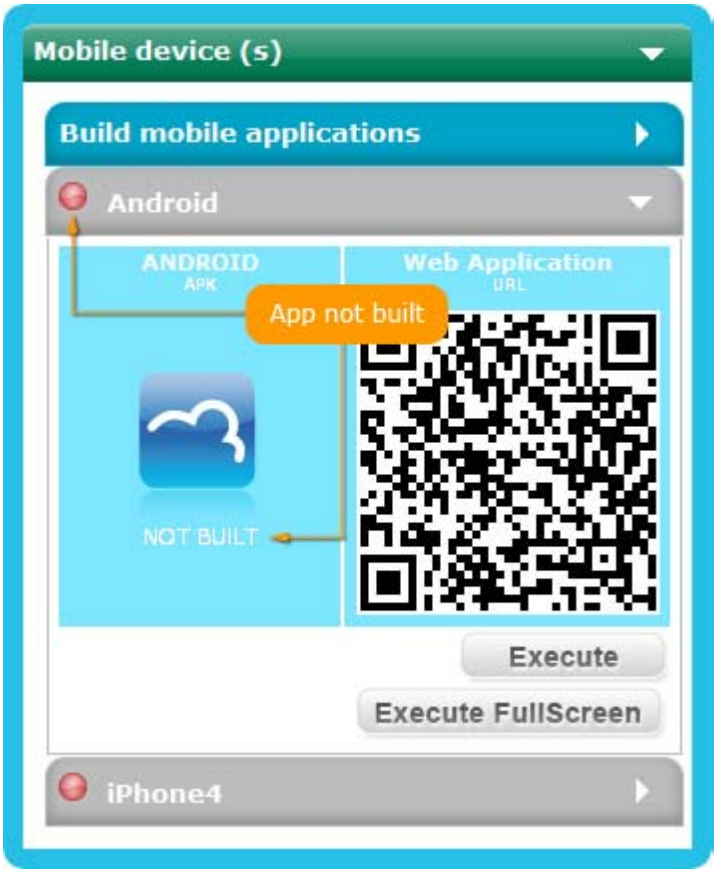


Figure 2 - 116: Android mobile device on test platform on Convertigo Cloud server

You can notice the red bullets appearing in front of each device, meaning that the mobile application for this device is not yet built. In the device panel, you also see the Convertigo Cloud image with `NOT BUILT` message.

3 Expand the **Build mobile applications** panel:

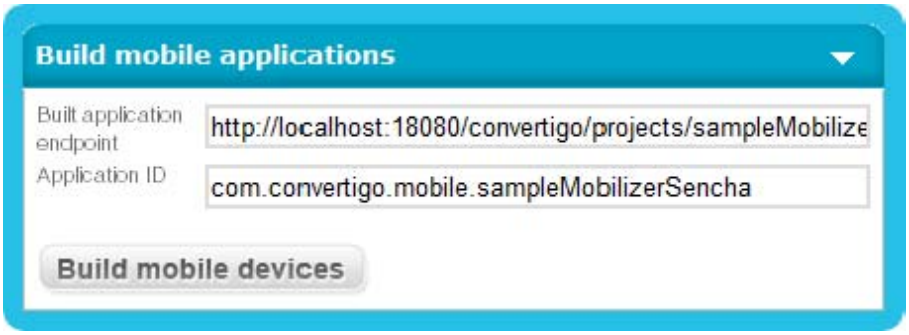


Figure 2 - 117: Build mobile applications panel on Convertigo Cloud server

4 If necessary, edit the **Built application endpoint** URL and the **Application ID**. These fields are positioned by default:

- the application endpoint is set to the project opened in the test platform. If you are using your Convertigo Studio project, configure here the URL pointing to the project deployed on a Convertigo Cloud server (private Convertigo Cloud or trial Convertigo Shared Cloud).

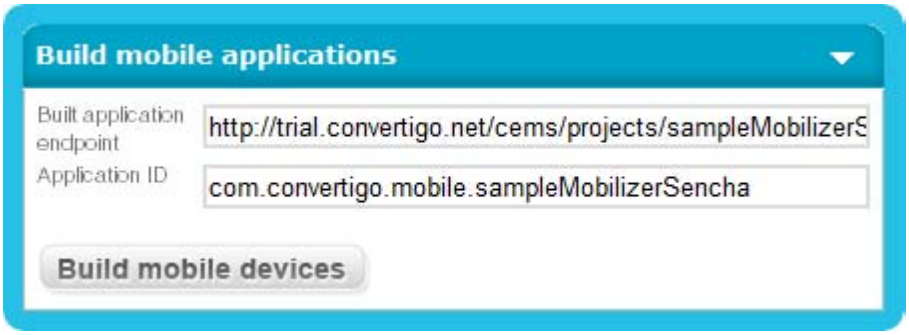


Figure 2 - 118: Editing Build application endpoint to Convertigo Cloud URL

- the application ID is set to a package name using the project name. Here, it doesn't contain any forbidden character so it doesn't need to be edited.

5 Click on the **Build mobile devices** button to launch the build on ***PhoneGap***.

The build process starts, the bullets switch to orange indicating the build is in progress on ***PhoneGap*** and a waiting icon appears:

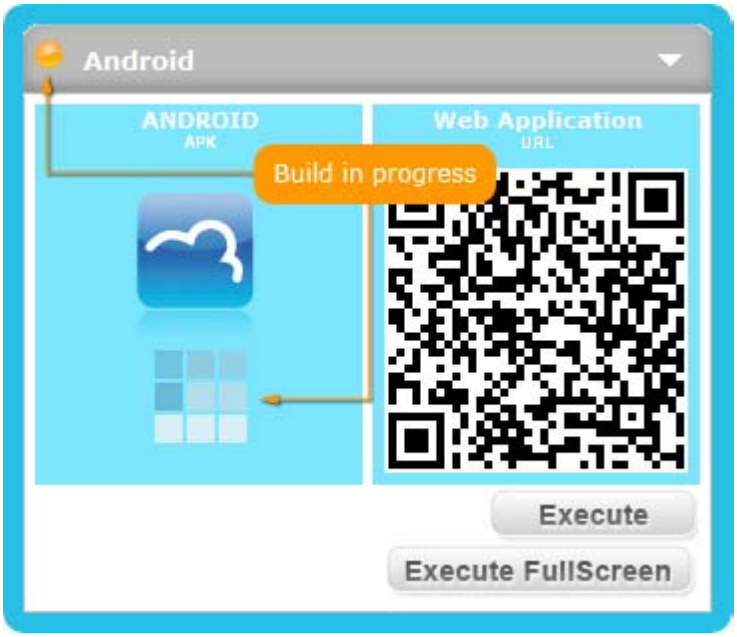


Figure 2 - 119: Applications build in progress on PhoneGap

When the build finishes, bullets color becomes green and a QR code is displayed in the left part of the panel:

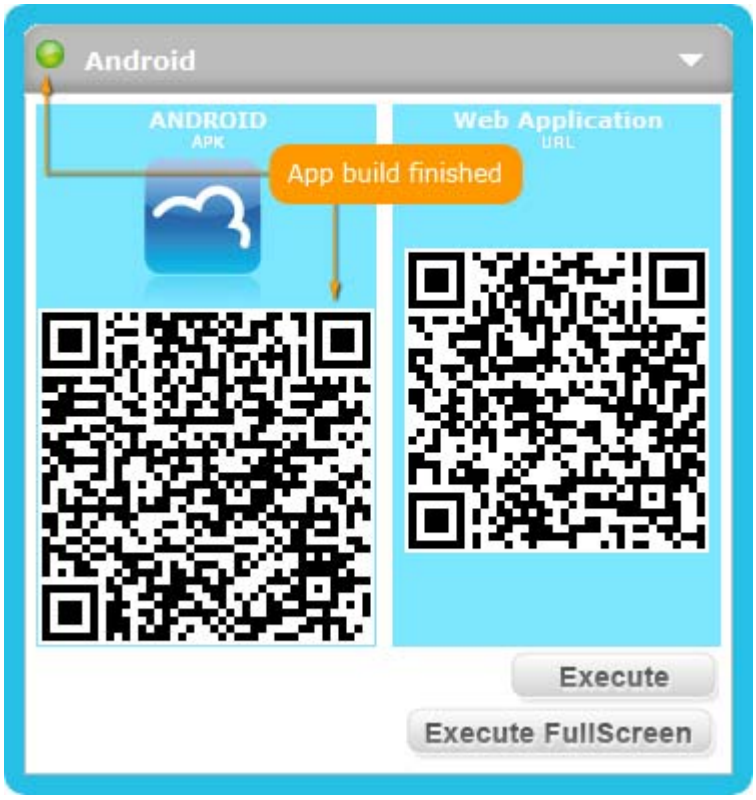


Figure 2 - 120: Applications build finished

*On current version of **PhoneGap build** platform, the iOS builds are not working directly. A manipulation can be performed directly on **PhoneGap build** interface to manage building the iOS application.*
*This procedure is detailed in the **Publishing to Apple's App store** section of the **Publishing Convertigo Mobilizer applications in application stores** Technical article on the Developer Network. You can find this article at: http://www.convertigo.com/en/how-to/blog-technical/entry/publishing-convertigo-mobilizer-applications-in-application-stores.html*

6 If you have an `Android` device, flash the QR code displayed on the left to download the application APK directly on your smartphone.

WHAT COMES NEXT?

Now that your mobile applications are built, you can install them on your mobile devices and publish them in application stores. To go further with publishing applications, see the **Publishing Convertigo Mobilizer applications in application stores** Technical article in Convertigo Developer Network at *http://www.convertigo.com/en/how-to/blog-technical/entry/publishing-convertigo-mobilizer-applications-in-application-stores.html*