**Programming Guide**

**CITADEL…on the move**

**Mobile Application Template: ‘Environmental Data’**

**Project co-funded by the European Commission within the ICT Policy Support Programme**

Table of Contents

[CITADEL…on the move 1](#_Toc357779215)

[1. Configuring the template 3](#_Toc357779216)

[2. Structure of the code 4](#_Toc357779217)

[2.1 HTML 4](#_Toc357779218)

[2.2 Javascript 4](#_Toc357779219)

[3. CITADEL sensor feed format 5](#_Toc357779220)

[3.1 Introduction 5](#_Toc357779221)

[3.2 JSON Example 5](#_Toc357779222)

[4. Translation mechanism 9](#_Toc357779223)

# Configuring the template

Every template comes with a set of configuration settings. These settings can be found in ‘Config.php’. They hold information about the city that the template is about, the use or not of a database as well as other configuration details that can change the appearance or the functionality of the template. Table 1 presents the available settings for the ‘Environmental Data’ template:

Table

|  |  |  |
| --- | --- | --- |
| setting | Default value | meaning |
| SERVERNAME | localhost | The name of the local web server instance. It should be changed to the IP address in order to be able to access the template from another device of the same network. |
| HTDOCS\_ROOT |  | The root directory of the web server. There is no standard path for this so it should be changed to match the current root directory. |
| BASE\_DIR | environmental-template/ | This is the root folder of the application. This should not be changed in no backend is used. |
| CLASSES\_DIR | php | This is the folder containing the php files. This should be not changed if the default folder structure is followed. |
| MAP\_CENTER\_LATITUDE | 53.483526 | The latitude of the centre of Manchester. Manchester is one of the two pilot cities that provide environmental sensor data. |
| MAP\_CENTER\_LONGITUDE | -2.23065399999996 | The longitude of the centre of Manchester. Manchester is one of the two pilot cities that provide environmental sensor data. |
| MAP\_ZOOM | 6 | The initial zoom level of the google map. |

# Structure of the code

The following sections provide a description and code examples of the way the mobile application templates are structured. The focus is on html, javascript and css files as they contain the implementation of the core functionality.

## HTML

The templates follow a Single Page Application (SPA) approach. This choice has been made in conjunction with the use of Jquery mobile for the appearance of the templates. There are three main ‘pages’ in the template, the map, the list and the details page. Each of them is actually a ‘div’ element with the ‘data-role’ attribute set to ‘page’. For example, the markup for the map page looks like this:

<!-- Home Page: Contains the Map -->

<div data-role=**"page"** id=**"page1"** class=**"page"**>

<header data-role=**"header"** data-posistion=**"fixed"** data-id=**"constantNav"** data-fullscreen=**"true"**>

<span class=**"ui-title"**>**Environmental Template**</span>

<div data-role=**"navbar"** class=**"navbar"**>

<ul>

<li><a href=**"#"** class=**"pois-nearme"** data-theme=**"b"**>**Near me**</a></li>

<li><a href=**"#"** class=**"pois-showall ui-btn-active"** data-theme=**"b"**>**Show all**</a></li>

<li><a href=**"#page2"** class=**"pois-list"** data-theme=**"b"**>**List**</a></li>

</ul>

</div><!-- /navbar -->

</header>

<div data-role=**"content"** id=**"map-container"**>

<div id=**"map\_canvas"** class=**"map\_canvas"**></div>

</div>

</div>

In the above code, you can also see the ‘header’ element which includes the navigation links. This must be present in every page that we want the navigation bar to be visible.

## Javascript

Much of the functionality provided in the templates is based on javascript libraries and the HTML5 javascript apis. The main actions involving javascript are:

* Get sensors’ feeds data by making ajax calls to Xively Api
  + function ‘getLiveFeeds
* Load the google map
  + function ‘initializeMap’
* Create Infobubbles for every sensor and add them on the map
  + function ‘setInfoWindowSensor’, function ‘addMarkers’
* Switch between the pages
  + ‘Event Handlers’ section in environmental-lib.js

Functionality inside JQuery mobile has also been used in order to assist with page switching. The code snippet presented below demonstrates such a case where a handler is added on the ‘click’ event of a button in the navigation bar. It is the ‘List’ button which presents the user with the Sensors in a list view.

/\* Click handler for the 'list' button \*/

$**(**'.feeds-list'**).**click**(*function*()** **{**

$**.**mobile**.**changePage**(**"#page2"**,** **{** transition**:** "none"**});**

**});**

# CITADEL sensor feed format

## Introduction

The mobile application “Environmental” template makes use of feeds coming via the XivelyAPI.

Xively uses a hierarchy of datatypes – feeds, datastreams and datapoints. A feed, or environment, is a context-specific collection of sensor-measured data at a particular geolocation. A *feed* is a data representation of an environment and its datastreams. Metadata associated with feeds allow for the specification of whether a feed is fixed or mobile, indoor or outdoor, etc. A *datastream* is a representation of an individual sensor within an environment. Datastreams are required to have unique alphanumeric IDs within the environment. Datastreams can also specify units (celsius for temperature, for example) as well as tags. *Datapoints* represent a single value of a datastream at a specific point in time.

An example of this data hierarchy in action is a city street (the environment) with traffic and noise sensors (the datastreams). A Xively feed could then be created with two datastreams with IDs “traffic” and “noise”, which could be assigned the units “cars per minute” and “decibels”. Datapoints at a given point in time could be “15” and “87”.

Feeds use a JSON representation which is described in details in the example below.

## JSON Example

The example presented in this paragraph provides an example of the JSON representation of data used by the Enironmental template. Data for the example has been taken by the Xively API.

The first elements, *id, title, private, tags, description*, *updated, created,**creator, location*, provide the available metadata of every sensor and then an array, *datastreams,* with the current sensor feed measurements follows. Every element of this array includes an *id, current\_value, at (when measured), max\_value, min\_value, tags,* and *a unit* arraywith *type, symbol* and *label.*

{

**"id":** 36075,

**"title":** "Current Cost London Home",

**"private":** "false",

**"tags":** [

"currentcost",

"electricity",

"london"

],

**"description":** "UPDATE: 20121128\r\n\r\nThis is now running on an android device through a debian chroot\r\n\r\nCurrent Cost Envi via an IONITX-P-E HTPC/NAS\r\n\r\n'Current Cost-to-Pachube' Java stability test. I'll probably need to move to a python/perl shell solution.",

**"feed":** "https://api.xively.com/v2/feeds/36075.json",

**"status":** "frozen",

**"updated":** "2013-05-31T08:13:07.068048Z",

**"created":** "2011-09-23T04:10:16.917567Z",

**"creator":** "https://xively.com/users/shaf",

**"version":** "1.0.0",

**"datastreams":** [

{

**"id":** "0",

**"current\_value":** "18.2",

**"at":** "2013-05-31T08:13:06.906470Z",

**"max\_value":** "21.4",

**"min\_value":** "11.3",

**"tags":** [

"celsius",

"degrees",

"temperature"

],

**"unit":** {

"type": "basicSI",

"symbol": "C",

"label": "Celsius"

}

},

{

**"id":** "1",

**"current\_value":** "00525",

**"at":** "2013-05-31T08:13:06.906470Z",

**"max\_value":** "17033.0",

**"min\_value":** "9.0",

**"tags":** [

"electricity",

"power",

"watts"

],

**"unit":** {

**"type":** "derivedSI",

**"symbol":** "W",

**"label":** "Watts"

}

},

{

**"id":** "loadavg",

**"current\_value":** "0.1",

**"at":** "2013-05-31T08:13:06.906470Z",

**"max\_value":** "50.61",

**"min\_value":** "-1.0",

**"tags":** [

"load"

]

}

],

**"location":** {

**"disposition":** "fixed",

**"name":** "Home",

**"exposure":** "indoor",

**"domain":** "physical",

**"ele":** "0",

**"lat":** 51.519853,

**"lon":** -0.12359600000002

}

}

The template recognizes and picks up the datastreams. For every datastream we get all its tags in order to specify what kind of measurement is to be represented. An implementation of this follows. The setDetailPageSensor function searches for the datastreams tag values “temperature”, “humidity”, “carbon dioxide”, ”noise”, “carbon monoxide”, “ozone”, “nitrogen dioxide”, or “electricity”. When one of these tags are found, a representative icon is displayed in front of the measured data value, in sensor’s details page. The code of the function that searches for them and initializes the contentTemplate variable follows:

***function*** setDetailPageSensor **(**feed**)**

**{**

/\* Get sensor specific attributes \*/

var contentTemplate =

"<div class='feed-data'>" +

"<div class='title'>" + feed.title + "</div>" +

"<ul>";

$.each(feed.datastreams, function(i, stream) {

var date = new Date(stream.at);

var measureDate = dateFormat(date, "dd/mm/yyyy, hh:MM:ss");

var image = "";

var text = "";

var iconLegend = "";

$.each(stream.tags, function(j, tg) {

tag = tg.toString().toLowerCase();

If the tags are found we also want to display a relevant icon before their values:

***switch*** (tag) {

***case*** "temperature":

image = "<img src='images/temperature2.png' alt='Temperature' />";

iconLegend = tag;

text = "";

***break;***

***case*** "humidity":

image = "<img src='images/humidity.png' alt='Humidity' />";

iconLegend = tag;

text = "";

***break;***

***case*** "carbon dioxide":

image = "<img src='images/icon-co2.png' alt='Carbon2' />";

iconLegend = tag;

***break;***

***case*** "noise":

image = "<img src='images/decibel.png' alt='Noise' />";

iconLegend = tag;

***break;***

***case*** "carbon monoxide":

image = "<img src='images/icon-co.png' alt='Carbon' />";

iconLegend = tag;

***break;***

***case*** "ozone":

image = "<img src='images/icon-o3.png' alt='Ozone' />";

iconLegend = tag;

***break;***

***case*** "nitrogen dioxide":

image = "<img src='images/icon-no2.png' alt='Nitrogen' />";

iconLegend = tag;

***break;***

***case*** "electricity":

image = "<img src='images/icon-power.png' alt='Electricity' />";

iconLegend = tag;

***break;***

***default*** :

***if*** (image != "") {

text = "";

}

***else*** {

text = "<div class='text'>" + tag + " " + "</div>";

image = "";

iconLegend = "";

}

***break***;

}

contentTemplate += "<li id='mylist'><span class='image-icon'>" + image + "</span>" + text + "<div class='showDetails'>" + iconLegend + "</div><span class='image-text'>" + stream.current\_value + " " + unit + "</span><div class='feedDetails'></div><div id='center-button'><a href='#' data-role='button' data-icon='feed-graph' onclick='historyGraphClick(" + feed.id + ",\"" + stream.id + "\"); return false;'>History </a></div><div class='valueDate'>"

+ measureDate + "</div><div class='feedDetails'></div></li>";

A tag “iconLegend” that actually describes what the icon depicts is also appears when the user clicks on the radio button which is placed in the beginning of the details page.

$("#isOff").click(function() {

$(".showDetails").hide();

});

$("#isOn").click(function() {

$(".showDetails").each(function() {

***if*** ($(this).html() != "")

{

$(this).show();

}

})

});

Changing the contents and markup of the ‘contentTemplate’ variable can result in different ways of visualisation that suit the application developer’s needs. Another example of easy layout customization is the content of the infobubbles. Infobubbles popup when a marker on the map is clicked. They may contain any information that is available about the selected sensor. The function that contains the template for the infobubbles in the ‘Environmental’ template is ‘setInfoWindowSensor’. As presented in the following code snippet, the chosen details for the infobubble are the title and the date, and the sensor first data value.

/\* Sets the content of the infoBubble for the given

\* feed

\*/

***function*** setInfoWindowSensor(feed)

**{**

***var*** date = new Date(feed.datastreams[0].at);

***var*** measureDate = dateFormat(date, "dd/mm/yyyy, hh:MM:ss");

***var*** contentTemplate =

"<div id='feedBubble'><a href='#page3' onclick='overrideDetailClick(\"" +

feed.id + "\"); return false;'>" +

"<div class='title'>" +

feed.title +

"</div>" +

"<div class= 'measureDate'>" + measureDate +

"</div>" + "<div class='value'>" + feed.datastreams[0].current\_value +

" " + feed.datastreams[0].unit.label +

"</div>" + "</a></div><div id='bubbleClose'><a href='' onclick='return overrideBubbleCloseClick();'><img src='images/close.png' width='25' height='25' alt='close' /></a></div>";

***return*** contentTemplate;

**}**

The history graph of a specific datastream is produced by Xively. A timeframe selection is available to users that can select to see historical data of the last 24 hours, 3 days or 14 days.

function historyGraphDateClick(feed\_id, datastream\_id, timerange) {

$("#graph").show();

***var*** startTimestamp = "";

***var*** now = new Date();

***var*** endTimestamp = dateFormat(now, "yyyy-mm-dd'T'HH:MM:ss'Z'");

***if*** (timerange == '24h')

{

startTimestamp = now.setDate(now.getDate() - 1);

}

***else if*** (timerange == '3days')

{

startTimestamp = now.setDate(now.getDate() - 3);

}

***else*** // 14 days, the max history xively offers

{

startTimestamp = now.setDate(now.getDate() - 14);

}

startTimestamp = dateFormat(startTimestamp, "yyyy-mm-dd'T'HH:MM:ss'Z'");

***var*** src = "https://api.xively.com/v2/feeds/" + feed\_id + "/datastreams/" +

datastream\_id + ".png?w=360&h=240&b=true&t=" + cachedfeedsArray[feed\_id].tags[0] +

"&start=" + startTimestamp + "&end=" + endTimestamp;

***var*** contentGraph = "<img src='" + src + "' alt='Graph loading...' />";

$('div > article p').html(contentGraph);

}

# Translation mechanism

The templates follow a simple mechanism to translate the user interface elements. A translation file named ‘translations.xx-XX.php’ contains a list of php variables that correspond to all the translatable strings of the user interface. The ‘xx-XX’ part of the filename must be replaced by the relevant language code and locale, e.g. en-GB for English (Great Britain) or fr-FR for French.

The ‘lang’ setting must be set accordingly when changing languages. The template will try to load the translation file with the given locale and will fall back to the default English version if the file is not found. A part of the English translation file for the ‘POIs in the city’ template follows:

**File: translations.en-GB.php**

**…**

$near\_me = “Near me”;

$show\_all = ”Show all”;

$info = “Info”;

$details\_title = “Details Page”;

…

The translation file should be in UTF-8 encoding