**Programming Guide**

**CITADEL…on the move**

**Mobile Application Template: ‘Crowd-sourced PoIs’**

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

Table of Contents

[CITADEL…on the move 1](#_Toc341960345)

[1. Configuring the template 3](#_Toc341960346)

[2. Structure of the code 3](#_Toc341960347)

[2.1 HTML 3](#_Toc341960348)

[2.2 Javascript 4](#_Toc341960349)

[3. CITADEL common POI format 4](#_Toc341960350)

[3.1 Introduction 4](#_Toc341960351)

[3.2 JSON Example 5](#_Toc341960352)

[3.3 Using tplIdentifier to customize the user interface of an application 6](#_Toc341960353)

[4. Translation mechanism 9](#_Toc341960354)

# 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 ‘POIs in the city’ template:

Table

|  |  |  |
| --- | --- | --- |
| setting | Default value | meaning |
| USE\_DATABASE | false | Indicates weather the template will use data from the database or not |
| DATASET\_FILE | POI\_gent.json | The name of the .json file that will be loaded if no database is used. |
| DATASET\_ID | 1 | The dataset id is created when a dataset is imported in the backend. The template comes with a prepopulated database which includes the same dataset as the default .json file. |
| 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. |
| DATASET\_URL | dataset.php | The url to the service that retrieves the dataset |
| 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 | citadel-crowd-sourcing-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 | 51.033261 | The latitude of the centre of the map. This should be the centre of the city in which the dataset refers to. |
| MAP\_CENTER\_LONGITUDE | 3.726488 | The longitude of the centre of the map. This should be the centre of the city in which the dataset refers to. |
| MAP\_ZOOM | 16 | The initial zoom level of the google map. |
| DB\_NAME | citadel | The name of the database. It should not be changed if the database dump is used to create the database |
| DB\_USERNAME |  | The username of a user with full rights to the database |
| DB\_PASSWORD |  | The password of the user |
| DB\_HOSTNAME | 127.0.0.1 | The default host name |
| DB\_PORT | 3306 | The default port |

# 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 php 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"**>

<a href=**"#info"** data-rel=**"dialog"** data-icon=**"info"** data-iconpos=**"notext"** data-theme=**"b"** title=**"Info"**>*&nbsp;*</a>

<span class=**"ui-title"**>**Crowd-sourced PoIs**</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:

* Read the POIs from the dataset
  + function ‘getPoisFromDataset’
* Load the google map
  + function ‘initializeMap’
* Create Infobubbles for every POI and add them on the map
  + function ‘setInfoWindowPoi’, function ‘addMarkers’
* Switch between the pages
  + ‘Event Handlers’ section in crowd-sourcing-lib.js
* Vote up/down a POI
  + #voteUpButton, #voteDownButton click handlers

Functionality inside JQuery mobile has been also 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 POIs in a list view.

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

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

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

**});**

## PHP

Php is used to support the backend functionalities of the template. Firstly, the configuration file is a php file which is included in the template and holds the values for the various setting of the template. The database communication as well as the exposure of the data through web services is also implemented in php. An object-oriented approach has been followed to represent the data model of the CITADEL common POI format (see Section 3). The database schema is presented in Figure 1:

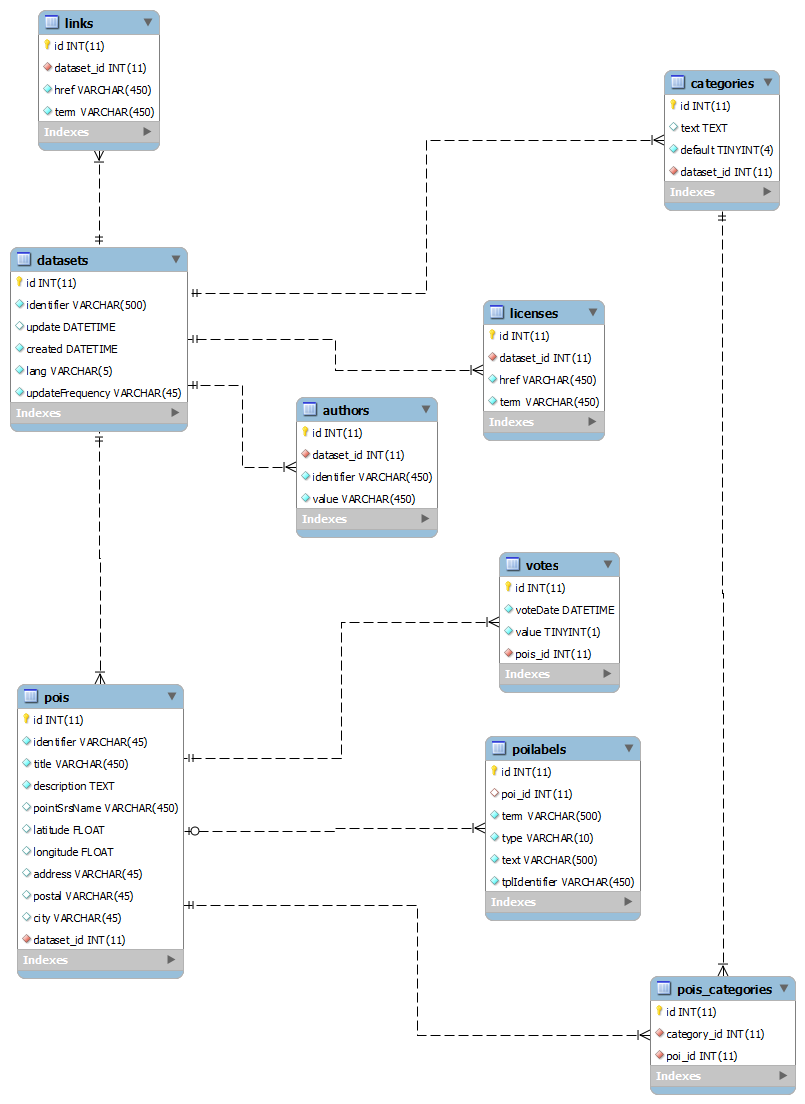


Figure : Database Schema

File ‘dataset.php’ contains the code which handles the loading of the dataset that is going to be used by the template. It checks whether the template is configured to use a database and loads the configured dataset id. If there is no database, the dataset will be loaded by the defined .json file.

When a user adds a new POI using the Citadel-Crowd-Sourcing-Template, the ‘insert.php’ script is called. This script is responsible to add the new POI in the database. The geocoder results contain a number of details for the given point, so if one wants to include them in the template he should also create the respective fields in the database table ‘pois’. The voting mechanism is implemented in the ‘voteManager.php’ script which is responsible for storing the users’ votes on the POIs. Furthermore, the HTML5 local storage feature is used in order to check whether a user has voted on a particular POI and prevent him from a second voting.

### Inserting a json dataset in the database

Another useful script that is available under the php folder is called ‘insertdataset.php’. This script reads the json dataset file from config.php and imports it in the database. If the import is successful, the new dataset Id is displayed in the screen. This dataset id should be put in the relevant setting (DATASET\_ID) inside the config.php so as to use this dataset.

# CITADEL common POI format

## Introduction

All the mobile application templates make use of a common data schema in order to ensure interoperability among different cities. The ‘Citadel-Crowd-Sourcing-Template’ template uses the CITADEL common POI format which has been derived from the [Editor’s draft of the Points of Interest Core of W3C](http://www.w3.org/2010/POI/documents/Core/core-20111216.html). The schema uses 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 templates. Data for the example has been taken by the publicly available dataset of POIs in the city of the city of Athens.

The first elements, *id, updated, created, lang, author, license, link* and *updatefrequency*, provide the available metadata of the dataset and then an array, *poi,* with the POIs follows. Every element of this array includes an *id, title, description, category, location* and a set of attributes*.*

The attributes are inside the *attribute* array and they contain information that is only available for POIs in the city. The ‘tplIdentifier’ values denote these poi-specific attributes and will be later used by the templates in order to provide a UI that is specific for POIs in the city.

{

"**dataset**": {  
 "**id**": "http:www.cityofathens.gr",  
 "**updated**": "20121030T09:38:21-5:00",  
 "**created**": "20121030T09:38:21-5:00",  
 "**lang**": "en-GB",  
 "**author**": {  
 "**id**": "http:www.cityofathens.gr",  
 "**value**": "City of Athens"  
 },  
 "**license**": { },  
 "**link**": {  
 "**href**": " http:www.cityofathens.gr ",  
 "**term**": "source"  
 },  
 "**updatefrequency**":"",  
 "**poi**": [  
 {  
 "**id**": "2768",  
 "**title**": "1 BASKETBALL COURT",  
 "**description**": "",  
 "**category**": [  
 "sports\_club"   
 ],  
 "**location**": {  
 "**point**": {  
 "**term**": "centroid",  
 "**pos**": {  
 "**srsName**": "http://www.opengis.net/def/crs/EPSG/0/4326",  
 "**posList**": "37.957243 23.721521"  
 }  
 },  
 "**address**":{  
 "**value**":"Cheldraich Street 15",  
 "**postal**":"11745",  
 "**city**":"Athens"  
 }  
 },  
 "**attribute**": [  
 {  
 "**term**": "Tel",  
 "**type**": "tel",  
 "**text**": "+302109216215",  
 "**tplIdentifier**" : "#Citadel\_telephone"   
 },  
 {  
 "**term**": "Nearest Metro",  
 "**type**": "string",  
 "**text**": "Neos Kosmos",  
 "**tplIdentifier**" : "#Citadel\_nearTransport"  
 },

{  
 "**term**": "url",  
 "**type**": "url",  
 "**text**": "http://www.breathtakingathens.com/node/2768",  
 "**tplIdentifier**" : "#Citadel\_website"  
 }

]  
 }

]

}

}

Data for POIs in the city that are created following the current schema will have a value in all fields that are marked with green color.

Fields marked with orange color are optional fields that could be used for a more complete presentation of the POI. They will be blank if not available.

Elements inside the ‘attribute’ array have been marked with blue color. These are the elements that will be different for every category of POIs. Moreover, if an Open Data provider possesses other pieces of information for a particular type of data, in this example ‘POIs in the city’, these could easily be added as a new element of the ‘attribute’ array. For example, adding the available sport to a specific poi of type ‘sports\_club’ (example above) would require the addition of the following element in the ‘attribute’ array:

"**attribute**": [

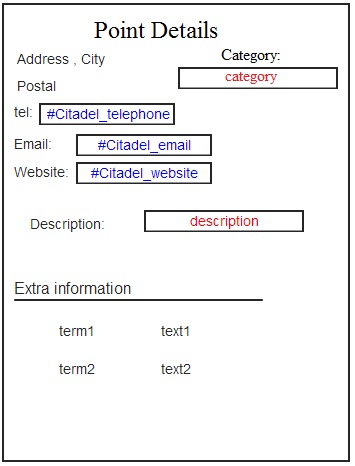
{  
 "**term**": "Available sports",  
 "**type**": "string",  
 "**text**": "Basketball",  
 "**tplIdentifier**" : ""  
 },…

]

The ‘tplIdentifier’ field would remain empty in this case so the nearest bus stops would be displayed in a list under the rest of the details for a parking lot.

## Using tplIdentifier to customize the user interface of an application

The element ‘tplIdentifier’ has a special meaning for the template, as described in the example of the previous paragraph. This element is a system-defined string that can be used to allow the templates to display attributes with different ‘tplIdentifier’ values in an aesthetically different way. If the attribute cannot be matched to one of the predefined ‘tplIdentifiers’ it will be displayed in a standard “list view” visualization, as seen in Figure 2.



Figure

The predefined values for ‘tmpIdentifier’ are:

1. "#Citadel\_website",
2. "#Citadel\_email",
3. "#Citadel\_parkType"
4. "#Citadel\_parkFloors"
5. "#Citadel\_parkCapacity"
6. "#Citadel\_telephone"
7. "#Citadel\_image",
8. "#Citadel\_eventStart",
9. "#Citadel\_eventEnd",
10. "#Citadel\_eventPlace”,
11. "#Citadel\_eventDuration",
12. "#Citadel\_openHours"
13. "#Citadel\_nearTransport"

The template recognizes and picks up the fields marked with a valid ‘tplIdentifier’. These can be displayed in predefined positions in the template’s interface, e.g. the category of a POI could be displayed in the upper right corner in red color. Depending on their type, they can also be rendered as telephone, email or website links. Terms that have not been matched with any of the available ‘tplIdentifiers’ would be just displayed as a simple list like ‘term1’ and ‘term2’ in Figure 1. The predefined list of ‘tplIdentifiers’ will be populated with fields coming from the original dataset provided by the pilot cities.

An implementation of the above scenario is followed in the Citadel-Crowd-Sourcing-Template. The setDetailPagePoi function searches for the poi-specific ‘tplIdentifier’ values ‘#Citadel\_website’, and ‘#Citadel\_ nearTransport’ in order to display them on top of the details page of a POI. The code of the function that searches for them and initializes the contentTemplate variable follows:

***function*** setDetailPagePoi**(**poi**)**

**{**

/\* Get the POI specific attributes\*/

***var*** website **=** getCitadel\_attr**(**poi**,** "#Citadel\_website"**).**text**;**

***var*** nearTransport **=** getCitadel\_attr**(**poi**,** "#Citadel\_nearTransport"**).**text**;**

***var*** contentTemplate **=**

"<h1>" **+** poi**.**title **+** "</h1>" **+**

"<div class='event-data'>" **+**

"<ul>"**;**

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

***if*** **(**website**)**

contentTemplate **+=** "<li><span>" **+** getCitadel\_attr**(**poi**,** "#Citadel\_website"**).**term **+** "</span>"

**+** website **+** "</li>"**;**

***if*** **(**nearTransport**)**

contentTemplate **+=** "<li><span>" **+** getCitadel\_attr**(**poi**,** "#Citadel\_nearTransport"**).**term **+**

"</span>" **+** nearTransport **+** "</li>"**;**

When we are done with these checks, we display all the rest attributes by taking their name (‘term’) and value (‘text’). We finally close the list of attributes and return the result to the main function.

***for*** **(**i **=** 0**;** i **<** otherAttributes**.**length**;** i**++)** **{**

contentTemplate **+=** "<li><span>" **+** otherAttributes**[**i**].**term **+** "</span>" **+** otherAttributes**[**i**].**text **+** "</li>"**;**

**}**

contentTemplate **+=** "</ul>" **+**

"</div>"**;**

***return*** contentTemplate**;**

**}**

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 POI. The function that contains the template for the infobubbles in the ‘POIs in the city’ template is ‘setInfoWindowPoi’. As presented in the following code snippet, the chosen details for the infobubble are the title and the category of the POI.

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

\* POI

\*/

***function*** setInfoWindowPoi**(**poi**)**

**{**

***var*** category **=** ""**;**

/\* Get the POI specific attributes\*/

***if*** **(**poi**.**category**.**length **>** 0**)** **{**

category **=** "<div class='category'>" **+**

poi**.**category**.**join**(**', '**)** **+**

"</div>"**;**

**}**

***var*** contentTemplate **=**

"<div id='poiBubble'><a href='#page3' onclick='overrideDetailClick(\"" **+** poi**.**id

**+** "\"); return false;'>" **+**

"<div class='title'>" **+**

poi**.**title **+**

"</div>" **+**

"<div class='address'>" **+** poi**.**location**.**address**.**value **+**

"</div>\n" **+** category **+**

"</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**;**

**}**

If the nearby transportation of the POI is what we want to show in the infobubble instead of the category, the code would have to change to (changes with bold font):

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

\* POI

\*/

***function*** setInfoWindowPoi**(**poi**)**

**{**

***var*** startdate **=** ""**;**

/\* Get the POI specific attributes\*/

***if*** **(**getCitadel\_attr**(**poi**,** "#Citadel\_nearTransport"**).**term **!=** ''**)** **{**

nearTransport **=** "<div class='nearTransport'>" **+**

getCitadel\_attr**(**poi**,** "#Citadel\_nearTransport"**).**term **+**

":" **+** getCitadel\_attr**(**poi**,** "#Citadel\_nearTransport"**).**text **+**

"</div>"**;**

**}**

***var*** contentTemplate **=**

"<div id='poiBubble'><a href='#page3' onclick='overrideDetailClick(\"" **+** poi**.**id

**+** "\"); return false;'>" **+**

"<div class='title'>" **+**

poi**.**title **+**

"</div>" **+**

"<div class='address'>" **+** poi**.**location**.**address**.**value **+**

"</div>\n" **+** startdate **+**

"</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**;**

**}**

As seen in the example, what is required to change is mainly the ‘tplIdentifier’ value. The name of the variable and the css class could remain the same but changing them improves readability of the code. Of course changing the css class also means that the appearance in the page will also change.

# 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.