CIMAM Mapping Interface

Design Document
User Manual
Version 1.0

Chirag Sangani

TABLE OF CONTENTS

INTRODUCTION

OVERVIEW

The Campus Information and Automation Management System (CIMAM) Mapping Interface is a framework that provides interfaces to integrate a sophisticated solution for displaying visual geographical or topological details regarding an entity and its environment. It provides a rich application programming interface (API) for integration, as well as a user friendly interface for administrative purposes. The end user interface is designed to be feature rich and flexible while conforming to design standards set by similar mapping solutions available on the internet.

FRAMEWORK DESIGN

The framework is designed to be flexible to support and adapt to the needs of different applications that may comprise an automation suite. Entities within a map are stored as objects in a database, bearing multiple attributes. Objects can be categorised by tagging: the system supports virtually an infinite number of tags. Tagging allows for a sophisticated search system, which can be utilized to emphasize specific objects in the generated interface.

Entities of a map are stored within the database as vector objects, their appearance stored in the Scalable Vector Graphic (SVG) format. A vector format allows for easy manipulation of the appearance of the object, such as its size, colour or position within the main map. Objects also carry descriptions and names, as well as custom attributes whose names and values can be set via the administrative interface, and accessed programmatically through a REST interface.

ADMINISTRATIVE INTERFACE

The administrative interface allows for addition or modification entities in the map. Addition is performed by uploading an SVG file, followed by automated parsing and object generation. Modification is performed by selecting particular entities from a map, which provides an interface to edit certain details, custom attributes, as well as delete the particular entry.

The frameworks flexibility and power makes it suitable for integration with automation modules, so as to provide visual information accompanied by textual information.

GETTING STARTED

OVERVIEW

The mapping framework is built on the Java EE framework, utilizing the EAR application format for deployment. Object data is maintained using a MySQL database, accessed via the MySQL connector for Java EE Persistence Unit. Deployment of the framework requires a server capable of handling EAR files: Glassfish or Apache Tomcat is recommended.

The client interface is designed to avoid dependence on any plugin. SVG rendering, as well as UI effects and animations, are accomplished through JavaScript libraries and frameworks. However, this puts a minimum requirement on the browser at the client: a minimum of Internet Explorer 8, Firefox 3.5, a WebKit browser such as Chrome or Safari, or Opera 10.

OBTAINING THE EAR PACKAGE

Deployment of the application requires generating a custom EAR package, since the server location in the network affects the functioning of the framework. It is also required to modify the persistence unit information to suit the particular database configuration. It is recommended to modify the source code through the NetBeans IDE, followed by building a distributable EAR package. The information to be modified is contained in the CIMAM object and the persistence.xml file.

ACCESSING THE VISUAL FRONT END

The framework provides a front end for end-users to access the mapping framework. This front end can be accessed through a web browser at:

```
#HOME#/CIMAM-war/
```

Where `#HOME#` is the base URL of the server.

The front end provides a view of the full map, along with the functionality to search for tags. The search query is a list of tags, separated by spaces. The result is the full map, with objects whose tags match the query highlighted. Additionally, separate tags can be highlighted with different colours by modifying the tag query to include after the tag the hexadecimal RGB value of the colour, the two being separated by a colon (:). An example query might look like:

```
foo bar:FF0000 ram:321F95
```

The map interface consists of panning and zooming controls on the bottom right corner. It also allows the interface to move between "floors", allowing access to information about multiple floors part of the same structure.

Each object visible in the interface is tagged with a name, if it is supplied within the database, and can be selected by clicking to reveal a popup with a description of the object. This description may contain arbitrary HTML code, and can be used to execute JavaScript code on the client's browser when the object is clicked. The database has a default description and colour associated with each object, though this may be overridden using an API call.

ACCESSING THE API

The Application Programming Interface is made available to integrating modules through REST calls: HTTP Get or Post messages to a particular URL with specific parameters that follow the API rules. The output of such a call is HTML code that can be embedded into webpages. The output is generated as per the specifications outlined in the API call.

Further details are presented in the Application Programming Interface section.

ACCESSING THE ADMINISTRATIVE INTERFACE

The administrative interface allows for addition and modification of entities within the database. Access to the interface can be obtained by pointing the web browser to the URL:

```
#HOME#/CIMAM-war/admin.jsp
```

Where `#HOME#` is the base URL of the server.

Access to the interface is guarded by an authentication scheme that allows only authorized personnel the permission to modify the database. A username as well as a password is required to access this interface. Information about allowed users is stored in a separate table.

Further details are presented in the Administrative Interface section.

APPLICATION PROGRAMMING INTERFACE

OVERVIEW

The API can be accessed by posting an HTTP GET or POST request to

`#HOME#/CIMAM-war/XMLInterface`

Where `#HOME#` is the base URL of the server.

There is only one parameter that is required to be passed, "q". The value of this parameter is an XML request as per the following specification.

XML FORMAT

```
<request>
	<format>
		#REQUEST_FORMAT#
	</format>
	<query>
		#REQUEST_QUERY#
	</query>
	<interface>
		#INTERFACE_PARAMETERS#
	</interface>
</request>
```

REQUEST FORMAT

Name:

`#REQUEST_FORMAT#`

Value:

`TagsAndHighlighting`

REQUEST QUERY

Name:

`#REQUEST_QUERY#`

Value:

```
<tagList>
	.
	<tag>
```

```
            <name>
                  #TAG_NAME#
            </name>
            <highlight>
                  #HEX_CODE#
            </highlight>
            <description>
                  #DESCRIPTION#
            </description>
      </tag>
      .
      .
</tagList>
```

Note that the highlight and description fields are optional. The tag name must contain an alphanumeric value.

INTERFACE PARAMETERS

Name:

```
#INTERFACE_PARAMETERS#
```

Value:

```
<size>
      <width>
            #WIDTH#
      </width>
      <height>
            #HEIGHT#
      </height>
      <edit>
            #TRUE_OR_FALSE#
      </edit>
</size>
```

Note that both width and height must be integers. The value of edit must be "false" at all times, except when making an API call that wishes to display options for editing an interface. Note that this call will work only if exists a frame named "CIMAMMapEditFormFrame" in the same webpage in which the map is displayed. Additionally, regardless of the fact that edit mode is on or off, the interface for editing objects will not work unless the user is logged in as an administrator. This call is used exclusively within the "edit.jsp" page of the administrative interface.

ADDITIONAL NOTES

1. If you wish to pass an ASCII string containing special characters as the contents of a tag, use `<![CDATA[#CONTENT#]]>` to prevent `#CONTENT#` from being interpreted as XML code.

   Example: `<description><![CDATA[<div>Hello!</div>]]></description>`

2. There should not be any unnecessary white space in the actual query passed to the user, since this is interpreted as part of the XML construct, and will result in an invalid query error.

EXAMPLE XML REQUESTS

1. The following request requests a map with no special highlighting, with a width of 800 pixels and a height of 600 pixels:

```
<request>
      <format>
            TagsAndHighlighting
      </format>
      <query>
            <tagList>
            </tagList>
      </query>
      <interface>
            <size>
                  <width>
                        800
                  </width>
                  <height>
                        600
                  </height>
            </size>
      </interface>
</request>
```

   Note that, while sending the actual query, it is required to remove any unnecessary whitespace, the presence of which will render the query as being interpreted as malformed.

2. The same query above, but without the unnecessary whitespace:

```
<request><format>TagsAndHighlighting</format><query><tagList></tagLi
st></query><interface><size><width>800</width><height>600</height></
size></interface></request>
```

3. The following request will highlight all objects tagged with "foo" in colour red, and will show a description of "bar" in the popup window when clicked.

```
<request>
      <format>
            TagsAndHighlighting
      </format>
      <query>
            <tagList>
                  <tag>
                        <name>
                              foo
                        </name>
                        <highlight>
                              FF0000
                        </highlight>
                        <description>
                              bar
                        </description>
                  </tag>
            </tagList>
      </query>
      <interface>
            <size>
                  <width>
                        800
                  </width>
                  <height>
                        600
                  </height>
            </size>
      </interface>
</request>
```

Once again, it is recommended to remove all whitespace before submitting the actual query.

RESPONSE UTILIZATION

The query access, when successful, returns HTML code which may be embedded in a web page. The simplest method to accomplish this is through the means of iframes: embedded frames within an HTML page. If the URL of the server is #HOME#, and the XML query is #XML#, then a map may be embedded by inserting the following code into the document where the map is to appear:

```
<iframe src="#HOME#/CIMAM-war/XMLInterface?q=#XML#" width="800"
height="600" scrolling="no" style="border:none">
```

Note that the above method is not guaranteed to work correctly, since it performs an implicit GET request. GET requests are URL encoded, and since URLs are typically supported only to a particular length, it is recommended that the request be performed by requesting a POST action. Such a method would involve AJAX, however, and is beyond the scope of this document. Additionally, it is advisable to URL-Encode the query if a GET request is to be performed, in order to prevent misinterpretation of the request.

ATTRIBUTE API

Each object can store multiple custom attributes: key-value pairs of metadata. These attributes may be modified using the administrative interface, but may be accessed via a REST API. The URL for obtaining these attributes is:

```
#HOME#/CIMAM-war/GetAttrib
```

Where `#HOME#` is the base URL of the server.

Required parameters:

`uid`: Unique Identifier of the object whose attributes are desired. The `uid` of any object may be obtained from the administrative interface.

Optional parameters:

`attrname`: The name of key of the desired attribute.

A single call may contain multiple `attrname` parameters or none at all. If only the `uid` is passed, all attribute key-value pairs are returned, else, only those that match the request are returned. An object may have multiple values with the same key: in that case, all the values will be returned.

The output is a bunch of key-value pairs, one on each line, in the following format:

```
key:value
```

ADMINISTRATIVE INTERFACE

OVERVIEW

The administrative interface allows for addition and modification of entities within the mapping framework database.

ACCESS CONTROL

The administrative interface comprises of the main administrator page, pages for addition and modification of the map. Access to all pages is restricted by a session-based login interface.

Login credentials are stored in the database, in the tables named "admin". Both the username as well as the password are stored as a hash code. Currently, the hash code used is the Java `hashcode()` function for strings. Additional credentials may be stored in the database as and when necessary.

A session is identified by a cookie stored in the client's browser. This name of this cookie is an authentication key, set to a randomized string to minimize the probability of a dictionary attack. The value of the key is also taken to be a randomized string. Presence of this key-value pair indicates that an administrator is logged in, though the user ID of the user cannot be determined as of yet. All cookie-related code is contained in the com.iitk.cimam.cookies package. The length of a session is currently set to one hour.

MAIN PAGE

The main page of the administrative interface provides two options: to add an entry as well as to modify existing entries. Other options include returning to the home page as well as logging out.

ADDITION OF NEW OBJECTS

The administrative interface supports the upload of an SVG file for adding new entities. This interface is accessible through the main page of the administrative interface.

The SVG format encloses information about vector objects in an XML document. A vector object may be of multiple types: such as rectangles, paths, polylines, and others. Currently, the interface supports SVG files containing the aforementioned three objects: files containing other types of objects will be rejected.

Along with the type of the object, the SVG format also stores other information, such as the location, width and height in the case of a rectangle, or the path string in the case of a path

or a polyline. The attributes mentioned are required for the upload interface to accept an SVG file.

The upload interface does not perform a thorough check on the XML document for compliance with the SVG standard, although it performs a check whether it complies with the XML standard. It requires one enclosing `svg` object, failing which it will deem the document as invalid. Subsequently, it performs a Document Object Model (DOM) tree search of the XML file for `path`, `polyline` and `rect` objects. These objects are then converted into paths, and stored as separate entities in the database. They are also tagged as "untagged", and names as "Unnamed". A successful upload redirects the user to the modification interface, where the user may populate the entities with metadata.

MODIFICATION OF OBJECTS

The modification interface allows a user to change the metadata associated with a particular object. Access to the interface may be obtained from the main page of the administrative interface.

The modification page shows a map, with objects tagged as `untagged` highlighted in red. A user may click an object to show the description pop-up. The pop-up window, along with a close button, now also contains an edit button, signified by a "+" sign. Clicking this button will open a form in a frame below the map, where a user may observe the object's `uid` or edit the name, tag list, highlight colour, offset as well as description of the particular object. A user may also edit attributes of the object by clicking the "Edit Attributes" link.

Validation of the form is performed on the server side.

FRAMEWORK ARCHITECTURE

OVERVIEW

The entire framework comprises of object modules that perform operations that, in combination, fulfil the requested query. Broadly, these modules can be classified as modules part of the Enterprise Java Bean (EJB) group, and those part of the Web Application Resource (WAR) group.

ENTERPRISE JAVA BEAN (EJB) GROUP

Modules contained within this group contain business logic, i.e. they perform actions independent of the query interface. These actions include parsing and validation of the query, database access and response generation. They may also raise exceptions in the case error states. The modules are enclosed by Web Application Resource modules, who then relay the appropriate responses to the user.

WEB APPLICATION RESOURCE (WAR) GROUP

The modules contained within this group expose the REST API to the programmer, the back end of the administrative interface provided to the authorized user, as well as the user interface presented to the end user. Subsequently, these modules concern themselves with the details of socket communication, extracting the request query from the HTTP request, invoking the appropriate business logic contained within the EJB group, collecting responses, catching exceptions, and returning them to the user in the appropriate format. The group also consists of the HTML and JSP pages and Javascript libraries that make up the administrative and user interfaces.

OBJECT MODEL

OVERVIEW

Figure 1: Object Model of the CIMAM Mapping Framework shows a block-level diagram that describes the major modules and the interactions between them. The objects have been shown classified as either EJB or WAR modules, and the package to which they belong is mentioned as well.

DIAGRAM

Refer Figure 1: Object Model of the CIMAM Mapping Framework.

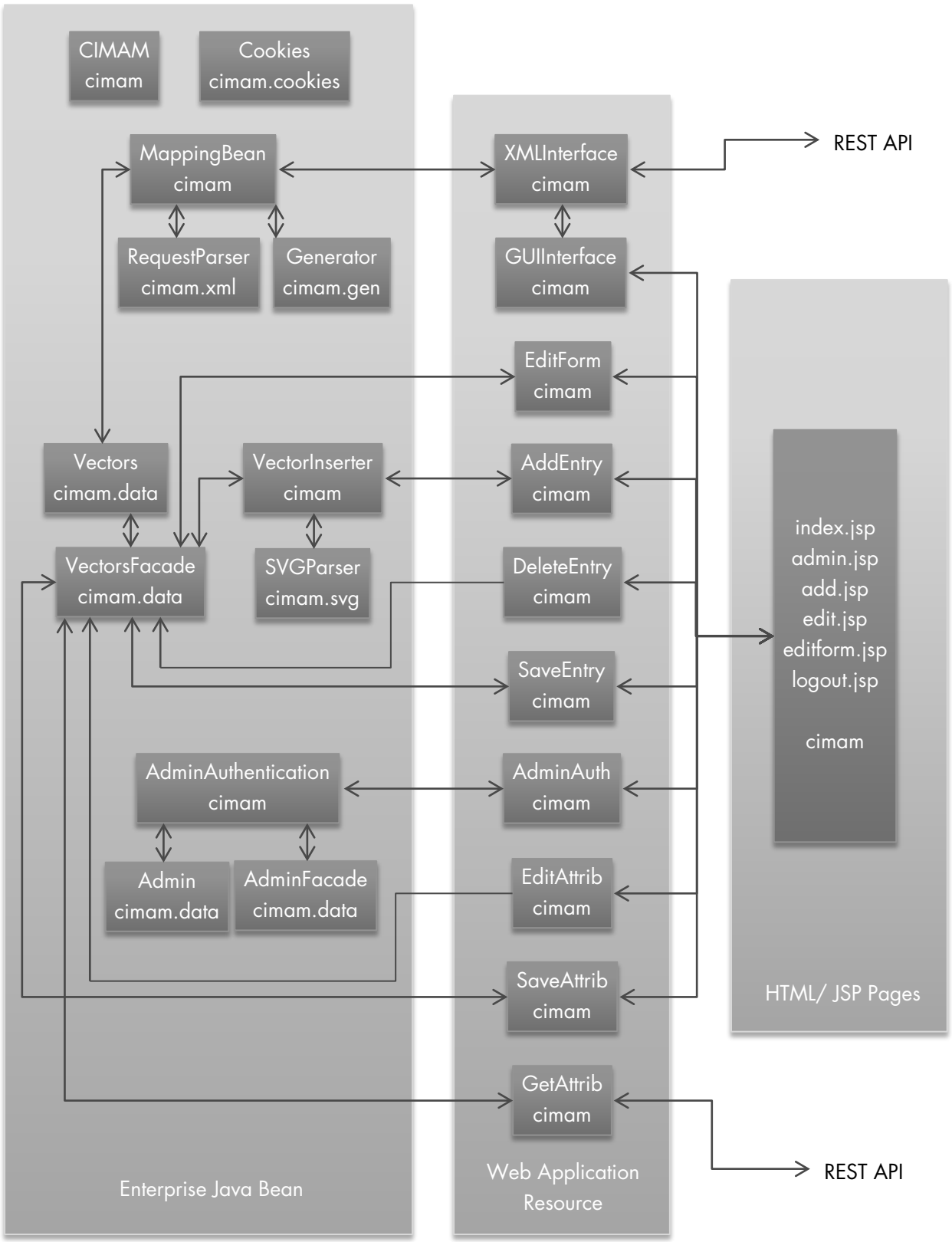

Figure 1: Object Model of the CIMAM Mapping Framework

MODULES

A brief description of each module follows:

EJB: cimam: AdminAuthentication

This module provides methods of validating authentication attempts. It accepts a username, a password, and a handle to an AdminFacade, and returns whether the username-password combination is valid or not.

EJB: cimam: CIMAM

This module contains framework-level static information, such as the base address of the server host

EJB: cimam: MappingBean

This module provides the logic to parse XML requests, obtain the necessary data from the database, and generate the required output, which it returns to the invoking object. It accepts an XML request in a string format, and a handle to a VectorsFacade. It returns the HTML code to be embedded in a string format. It instantiates the RequestParser and the Generator.

EJB: cimam: VectorInserter

This module accepts a Vectors object for insertion into the database. It accepts a Vectors object and a handle to a VectorsFacade.

EJB: cimam.cookies: Cookies

This module contains logic for identifying cookies and obtaining cookies for embedding inside the client web browser for an authenticated session.

EJB: cimam.data: Admin

This module is an object-oriented description of the admin table in the database.

EJB: cimam.data: AdminFacade

This module provides logic to search, add, edit and delete objects from the admin table in the database.

EJB: cimam.data: Vectors

This module is an object-oriented description of the vectors table in the database.

EJB: cimam.data: VectorsFacade

This module provides logic to search, add, edit and delete objects from the vectors table in the database.

EJB: cimam.gen: Generator

This module generates the requisite HTML code for the end user interface. It comprises of logic as well as static HTML files. It accepts a list of parameters for altering the output. Accordingly,

it loads the static files, modifies demarcated strings with changes specific to the incoming query, generates a finalised HTML code and returns it in a string format.

EJB: cimam.svg: SVGParser

This module provides functionality to read, parse and validate SVG files, as well as extract specific information from them.

EJB: cimam.xml: RequestParser

This module provides functionality to read, parse and validate XML requests, as well as extract specific information from them.

WAR: cimam: AddEntry

This module accepts multi-part form data, extracts parameters and SVG data from it, performs validation, and invokes the VectorInserter.

WAR: cimam: AdminAuth

This module accepts login credentials, performs validation, invokes the AdminAuthentication module and returns the result.

WAR: cimam: DeleteEntry

This module accepts the UID of the entry to be deleted and performs the action directly.

WAR: cimam: EditAttrib

This module accepts the UID of an object, and redirects the user to a form page that allows modification of the attributes of that object.

WAR: cimam: EditForm

This module accepts the UID of an object, and redirects the user to a form page that allows modification of the properties of that object.

WAR: cimam: GUIInterface

This module accepts the query string of the user front end, parses and validates it, and redirects the user to a map interface by invoking the XMLInterface module.

WAR: cimam: GetAttrib

This module directly implements the API for obtaining the attributes of a particular object.

WAR: cimam: SaveAttrib

This module accepts modified attributes for a particular object, performs validation, and performs the operation directly.

WAR: cimam: SaveEntry

This module accepts modified properties for a particular object, performs validation, and performs the operation directly.

WAR: cimam: XML Interface

This module accepts the XML request from an API query, invokes the MappingBean module and returns the result.

FUNCTIONAL DETAILS

The signatures of the functions present in each module are available in the JavaDocs for the EJB and the WAR groups.

DATABASE SCHEMATICS

OVERVIEW

This section defines the schemas of the various tables present in the CIMAM mapping framework, and provides additional information about the attributes.

ADMIN TABLE

Name	Type	Description
username	Integer	Hash code of the name of the user
password	Integer	Hash code of the password of the user

VECTORS TABLE

Name	Type	Description
UID	Integer	Unique identification number of the entry
tagList	TEXT	Comma separated list of tags of the entry, each tag must be an alphanumeric value
SVG	TEXT	Path string describing the shape of the object as per the SVG standard
x	REAL	X-coordinate of the object
y	REAL	Y-coordinate of the object
z	REAL	Floor of the object
width	REAL	Width of the object (not used)
height	REAL	Height of the object (not used)
description	TEXT	HTML description of the object
highlight	TEXT	Highlight colour of the object. Must be a 6-digit hexadecimal number
name	TEXT	Name of the object.
attributes	TEXT	Semi-colon separated list of attribute key-value pairs in the format "`key:value`"

OPERATIONAL FLOWS

OVERVIEW

This section covers some of the flows present in the framework. In particular, it addresses the flow of making an API call (which includes the operational working of the framework), the flow of adding new objects and the flow of editing objects.

API REQUEST

The API request flow begins with a client making a HTTP GET or POST request. This request is received by the XMLInterface module, which retrieves the query from the request. The query is passed on to the MappingBean, along with a handle to a VectorsFacade object.

The MappingBean invokes a RequestParser to parse and validate the request, and extracts a list of tags to be highlighted along with custom descriptions and colours, if any, as well as other interface parameters. It then invokes the Generator and passes this list of tags to it, along with the handle to the VectorsFacade object.

The Generator extracts all objects from the database using the VectorsFacade object and generates the HTML output using the static files stored along with it. This output is returned to the MappingBean, who then returns it to the XMLInterface, who returns it to the user

ADDITION OF OBJECTS

Addition of objects begins with their design in Vector editing software such as AutoCAD or Illustrator. The object is to be saved in the SVG format. Log in to the administrative interface, and select the "add new entry" option. Select the file to upload, and the initial offset. If the file is valid, it will be accepted and the user will be redirected to the editing interface, with new objects highlighted in red present in floor 0.

MODIFICATION OF OBJECTS

Modification of objects can be performed in the administrative interface. Log in, and select the "edit entries" option. Select the object to edit, and click the "+" in the popup window. Edit the properties in the form below, and click on save.