TextService

OMiLAB Technical Documentation

Software version: v0.3.3

Contents

Revision History

Revision	Date	Author(s)	Description
1.0	12.04.16	D Goetzinger	Created Document

1 Service Details

Prerequisites for the user

- None

Prerequisites for administrative user

- Basic HTML and CSS knowledge is beneficial

Prerequisites for the service operator

- SQL
- HTML
- CSS
- Tomcat server administration skills
- Familiarity with the concepts of the OMiLAB Microservice Infrastructure

Dependencies

- ActivityLoggingService
- FileManager
- MySQL

Frameworks

- Spring Boot 1.2.5.RELEASE and related
- CKEditor 4.4

Summary
The TextService provides the ability to display and edit static information like text or images in the HTML format. It is possible to edit HTML directly or use a user-friendly WYSIWYG editor. Theoretically all content, that can be represented through HTML, is supported as input for the TextService. The TextService is backed by a generic schema, consisting of elements and pages, that can be rearranged dynamically and is used to classify basic meaning to the content.

2 User Manual

The main use case of the TextService consists in displaying static content, mainly in the form of text and images.

In the administration menu of the TextService static text and images can be easily embedded by means of a WYSIWYG editor. Although it is not the focus of this service, its input is not restricted to text and images, but virtually any content can be displayed. In the upper left corner, there is button, which enables the administrator to enter plain HTML. Additionally several functions are available to facilitate the embedding of YouTube videos or creating tables. As source for images or links the OMiLAB FileManager can be browsed. Further documentation on how to use the editor in detail can be found at: `http://ckeditor.com/tmp/nature/`.

All input made by the administrator is automatically sent to the TextService and saved to the database. If there are any issues saving the content, it will be indicated by the green box, saying *"All changes are saved!"* turning red and giving more information about the error. In this case save all your changes somewhere else, i.e. copy and paste them to a text editor, reload the page and try it again. If the problem persists contact the service provider.

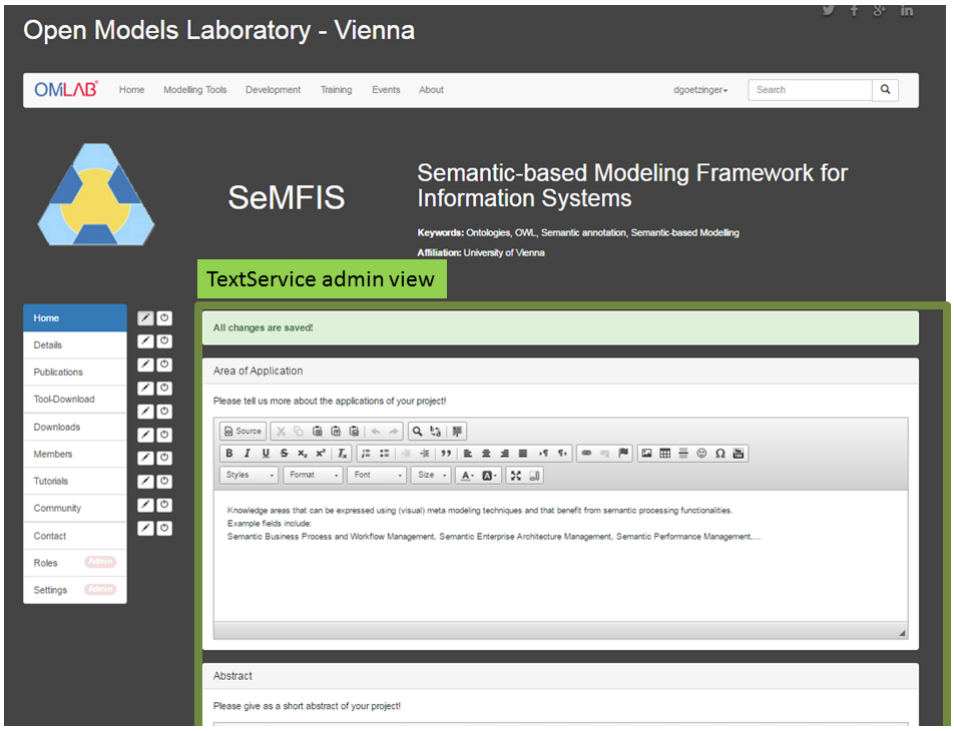

Figure 2.1: Administrator's view of the TextService

3 DevOps Manual

3.1 Installation

The installation of the TextService may be performed similar to all other services. It can be built from source using maven as build system. The according target to create a deployable package, is *"package"*. The profile *"distribution"* provides a well-documented configuration with sane default values. So the command to build the TextService using the command line interface can be *"mvn -Pdistribution package"*. This will create a deployable war archive in a new generated *"target"* folder.

Before a deployment on the application server is possible, the service has to be configured for the specific environment, which is configured in the file *"application.properties"*. If the configuration-section of the application-distribution.properties has not already been edited before building, the file can be found in the war archive at *"/WEB-INF/classes/application.properties"*.

```
# Address where the application is externally reachable
# (including application context, but without trailing
# slash)
app.url=http://www.omilab.org/textservice

# Credentials for database access
spring.datasource.url = jdbc:mysql://localhost:3306/text
spring.datasource.username = text
spring.datasource.password = password

# URL where the endpoint of ActivityLoggingService is
# reachable (without trailing slash)
omilab.activity = http://localhost:8080/logging
# SID (used for ActivityLoggingService) has to be
# unique across all services and is
# used to identify the service
omilab.sid = TEXT-VIE-01

# Define location of the OMiLAB Repository (without trailing
# slash)
omilab.repository = http://www.omilab.org/repo/filemanager
```

What is left to do after the configuration, is the preparation of the database. When the option *"spring.jpa.hibernate.ddl-auto"* is set to *"validate"* the service expects to find valid schema at the database given and will exit, if it is not present. In order to create the necessary schema the file *"schema.sql"* has to be imported to the configured database. This file contains already some standard configuration for the TextService, i.e. pages with the endpoint *"home"*, consisting of the elements *"Abstract"* and *"Area of Application"*, and the endpoint *"details"*, consisting of only one *"Details"* element are already present.

After the service is configured and the database schema is imported, the service may be deployed on the application server. The service has been tested extensively with Apache Tomcat 7 and JRE 7, as well as MySQL 5.5.

3.2 Configuration

The main concepts the TextService is using are an Element, a Page, the Instance, and the definitions of them. This enables a more dynamic arrangement of the elements and pages later on and a basic assignment of the semantics of an element.

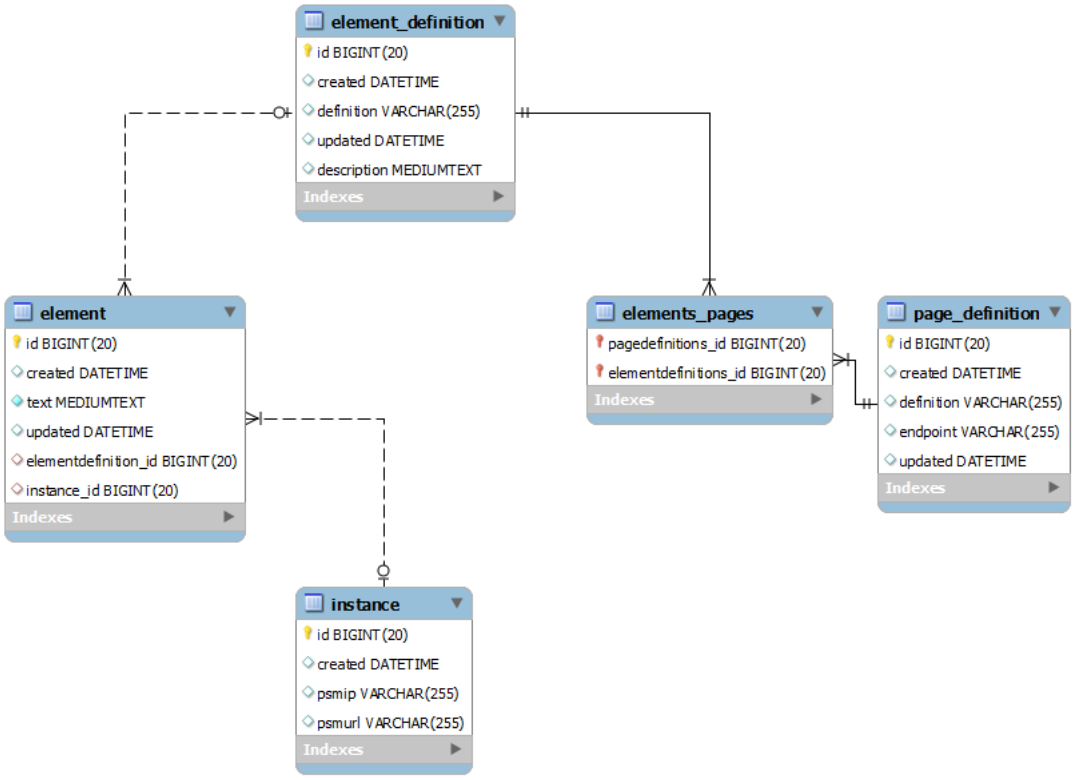

Figure 3.1: TextService database scheme

In the table *"element_definition"* the different types of elements are stored. They are the abstract definition of the white boxes visible in figure **??** and respectively figure 2.1.

The *"definition"* in this case is a simple internal value denoting the meaning of the entry. It will be visible as headline above the according element. The *"description"* will only be visible to the administrator and give him some direction on what to enter in the according element.

The *"page_definition"* again contains a similar *"definition"* column, bu t also a column for the *"endpoint"*. This one is needed so that a menu item at the PSM can be matched to the according page. So, for each page an according endpoint configuration, matching the *"endpoint"* column, an endpoint has to be added at the PSM configuration.

The relation between these two tables, is defined using *"elements_page"*. Based on this third table, it is specified which page contains which elements, based on its IDs. By means of this design it is effortless to change this assignment on the fly. Finally the *"element"* table finally holds the actual text data of the TextService.

Example: The creation of a new page, which is reachable by the endpoint *"assignment"* and comprised of the elements *"Problem statement"* and *"Solution"*, looks like the following:

```
1 INSERT INTO page_definition (definition,endpoint,created)
    VALUES ('Assignment','assignment',CURDATE());
2 INSERT INTO element_definition (definition,description,
    created) VALUES ('Problem statement','Please enter your
    problem statement below.',CURDATE());
3 INSERT INTO element_definition (definition,description,
    created) VALUES ('Solution','Please enter the solution
    for the problem below.',CURDATE());
```

Based on the IDs that have been generated for the statements above, the elements have to be assigned to the according page, e.g.

```
1 INSERT INTO elements_pages (pagedefinitions_id,
    elementdefinitions_id) VALUES (1,1);
2 INSERT INTO elements_pages (pagedefinitions_id,
    elementdefinitions_id)VALUES (1,2);
```

Now the endpoint *"assignment"* can be used at the PSM configuration.

Sometimes no specific structure is required and database access is not feasible. In this case the TextService is also able to react dynamically to new endpoints, i.e. if there is an endpoint configured at the PSM, that is not present at the TextService. In this case a *"page_definition"* with exactly one *"element_definition"* will be created. The *"definition"* column in this case will contain the name of the endpoint, prefixed with the string "custom_". Furthermore there will be no heading of the corresponding elements. Whenever an instance is removed, the TextService will check, if there are such definitions, starting with the string "custom_", that are not used by any *"Element"*. If this is the case, it will take care of removing it automatically.

How the different database concepts are represented in the user interface of the service is illustrated in figure 3.2.

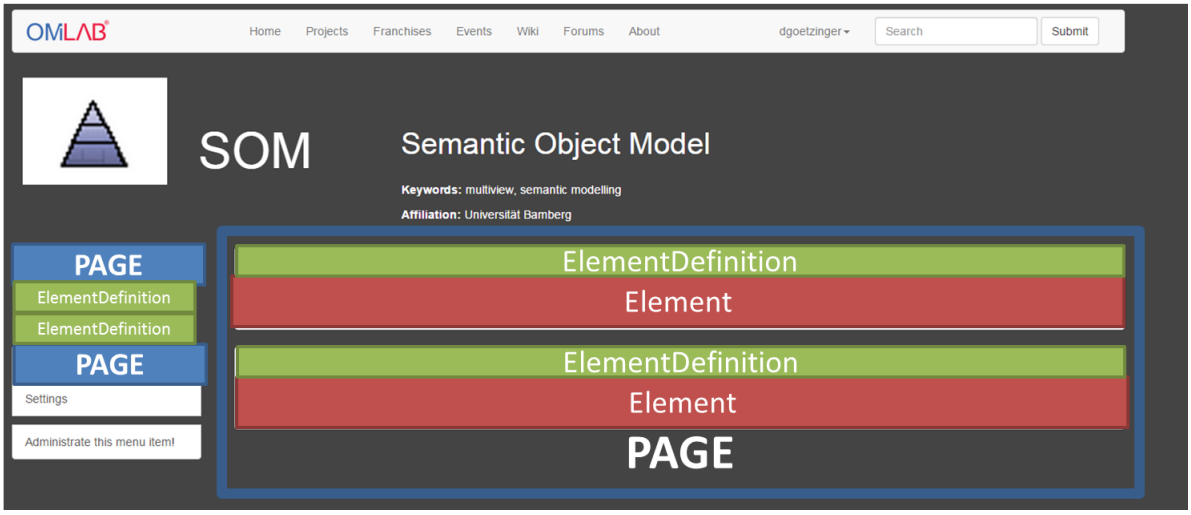

Figure 3.2: Display of TextService database concepts in the GUI

3.3 Interface and Implementation details

Generally the TextService assigns unique IDs to the elements, which are then used to save the modified text to the according elements.

Furthermore there is also the possibility to write to the elements of an instance directly based on their defined *"page_definition"*. This is especially useful, when some standard text should be assigned during the PSM Wizard Process. An example on how to set the content for the definition "Abstract" might look like following:

```
{
  "params": {
    "Abstract": "This is a standard message..."
  },
  "roles": ["PROJECT_OWNER"
  ],
  "username": "$USERNAME"
}
```

The auto-save features of the TextService is realised by recreating the form submit in JavaScript and firing on each trigger of the "change" event of the CKEditor. In order to know in the backend, whether the request was performed using this ajax method, or if it was the request to this service, a according GET-parameter is attached to the requested URL (ajax=true). This is particularly useful, to identify which changes have to be logged in the ActivityLoggingService. The TextService detects, if a request has been successfully performed by looking for the comment "TextService received request" in the response.

3.4 Logging Events

This section should take a closer look, on what events are sent to the LoggingService by the TextService. The creation and update of *"TextElements"* is logged.

1756	update	2016-03-31 07:51:34	Instance	44	dopplers	2
1757	create	2016-03-31 07:51:38	Element	1162	dopplers	2

Figure 3.3: Example log entries

4 Contact

4.1 Service Developer

David Götzinger

Research Group Knowledge Engineering
Faculty of Computer Science
University of Vienna

Währinger Straße 29
1090 Vienna
AUSTRIA

Email: dgoetzinger@dke.univie.ac.at