

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| **Document title** | **JAVASTENCIL** |
| **Document reference number** | **UEL -** |
| **Project name** | **UNISA e-Content 20140363** |
| **Unisa Department** | **Department of Economics** |

Table of Contents

1 Description of the component 5

2 How to Install The component 5

3 how to use the component 6

Document sign-off

This document has been seen and accepted by the following people:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Title | Date | Signature |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Compiler | Purpose | Change Summary |
| <yyyymmdd> | Name Surname | Document Created,  Internal Review,  External Review,  etc | Initial release,  Updates based on internal review,  Updates to section xxx,  etc |
|  |  |  |  |
|  |  |  |  |

# Description of the component

JAVA STENCIL

* What is the JavaStencil?
* Installation of the java stencil
* OUPUT Text
* OUPUT the GUI.
* Description and implementation of the Observer patter

## Description of the Java stencil

The Java stencil is the main container which is set up as main container and holding all the components developed in Java. The Stencil supports Tsugi implementation which is compatible with the LTI 2.0.

# How to install to the Java stencil

## Pre-Requisites:

Install Tsugi PHP and set it up:

<https://github.com/csev/tsugi>

This should setup all the database tables.

Using from pom.xml to set the depencies insert this code in your pom.xml file.

<dependency>

<groupId>org.tsugi</groupId>

<artifactId>tsugi-java</artifactId>

<version>0.1-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>org.tsugi</groupId>

<repositories>

<repository>

<id>ossrh</id>

<name>Sonatype</name>

<url>https://oss.sonatype.org/content/repositories/snapshots/</url>

<layout>default</layout>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

</repositories>

## Configuration

Databse Setup:

This is expecting that PHP Tsugi already is installed running and its database is created and available on localhost:8889 using the default account, password, and database name and that the tables already exist. If you want to change this, edit the file.

src/main/resources/tsugi.properties

Finally . You can clone the following repo.

<https://github.com/EonConsulting/JAVASTENCIL>

# How to use the stencil

## How to output GUI

We have provided two ways of getting a GUI. Either a form or a list. I will take you through how to add more as well.

-FORM

To create a form element, we need to start by creating the factory.

GuiFactory factory = GuiFactory.INSTANCE;

Now we need to specify which type of view we would like to use

Now we need to specify which type of view we would like to use. For form, we need to use the GUIEnum, FORM constant.

Gui gui = factory.createGui(GuiTypes.FORM.name());

And finally, we need to write the view with the following line of code:

gui.write();

-LIST

We will use the exact same code as above, except that now, we will use the GUIEnum, LIST constant to create the view.

To create a form element, we need to start by creating the factory.

GuiFactory factory = GuiFactory.INSTANCE;

Now we need to specify which type of view we would like to use. For List, we need to use the GUIEnum, LIST constant.

Gui gui = factory.createGui(GuiTypes.LIST.name());

And finally, we need to write the view with the following line of code:

gui.write();

## The Observer Pattern

3.2.1 Introduction

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

3.2.2 Implementation

Observer pattern uses three actor classes. Subject, Observer and Client. Subject is an object having methods to attach and detach observers to a client object. We have created an abstract class Observer and a concrete class Subject that is extending class Observer for you.

ObserverPatternDemo, our demo class, will use Subject and concrete class object to show observer pattern in action.

3.2.3 To make use of this pattern, we have made it easy for you.

1. Create a class that extends the Observer class in the below namespace

EonConsulting/JAVASTENCIL/tree/master/src/main/java/za/co/unisa/observerObjects

1. Make sure you have a constructor inside the class you created, which has a parameter subject of data type Subject. Within that constructor call the method attach(). The constructor should look like this ….

public observerObject(Subject subject) {

this.subject = subject;

this.subject.attach(this);

}

3. Call the update() method.

3.2.4 Using the pattern

Now, create the object like you would any other.

Subject subject = new Subject();

new observerObject(subject);

To attach the observers you wish to call you can use the below code.

this.subject.attach(this);

And finally, when you would like to call the observers, go ahead and execute the following line of code:

public void notifyAllObservers(){

for (Observer observer : observers) {

observer.update();

}

}

Appendix A

For Appendices use the style “UEL Appendix”