**PERSONALIZATION GUIDE**

1. Introduction 4

2. Source code 4

Download 4

Source code projects 4

Enforcement 5

3. Build the code 6

# Introduction

It is possible to personalize different features from the SLA core depending on the project where it is going to be used. Six different classes can be rewritten and configured in the SLA Core:

* Two of them from the enforcement; it is possible to change the way constrains are evaluated and the way the metrics are retrieved.
* Four of them for paring. It is possible to change the way Templates and Agreements are received and send. Parsers can be implemented to marshall and unmarshall data in json and xml formats.

# Source code

## Download

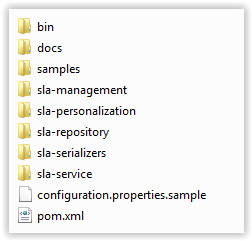
Follow the steps from the installation guide. There you will find the requirements for the SLA core. In this installation guide you will have the steps to download the project and for configure the machine for the database.

In this document it is also described the way to import the source code into eclipse.

## Source code projects

The project is made up of five main modules and can be found under <http://atossla.atosresearch.eu/svn/atossla/trunk> SVN

* SLA Repository
* SLA Management
* SLA Serializers
* SLA Service
* SLA Personalization



The **configuration.properties.samp**le that is placed in the parent directory has to be copied to **configuration.properties**. It is possible to find more information about it in the installation guide.

The SLA Personalization project is empty and should be used to implement the six classes that can be rewritten from the core. As it has been mentioned in the introduction, two groups of classes can be implemented:

* For the enforcement
* For parsing

The SLA Personalization project already imports the SLA Management, SLA Repository and SLA Serializers projects and will be included in the war generated to be deployed.

### Enforcement

TODO

PARSING

To implement a new parser, it doesn’t matter if it’s for and Template, an Agreement, for JSON or XML format, the class must extent from the eu.atos.sla.parser.IParser.

**public** **interface** IParser <T> {

/\*

\* getWsagObject receives in serializedData the object information in an xml, json or any other format

\* and must return the T object (eu.atos.sla.parser.data.wsag.Agreement or an

\* eu.atos.sla.parser.data.wsag.Template)

\*/

**public** T getWsagObject(String serializedData) **throws** ParserException;

/\*

\* getWsagAsSerializedData receives in serializedData the object information in an xml, json or any other

\* format and must return information following and xml in wsag standard.

\*/

**public** String getWsagAsSerializedData(String serializedData) **throws** ParserException;

/\*

\* getSerializedData receives in wsagSerialized the information in wsag standard as it was generated with

\* the getWsagAsSerializedData method and returns it in a xml, json or any other format

\*/

**public** String getSerializedData(String wsagSerialized) **throws** ParserException;

}

In the **configuration.properties** from theroot directory you must change the one of the following variables: parser.json.agreement.class, parser.json.template.class, parser.xml.agreement.class, parser.xml.template.class

Depending if it is parser.**json**… or pasrer.**xml**… the *serializedData* variable from the methods will be in json or in xml format.

Following example is the default agreement parser for JSON format

**public** **class** AgreementParser **implements** IParser<Agreement> {

/\*

\* getWsagAsSerializedData receives in serializedData json

\* and returs an Agreement object.

\*/

**public** Agreement getWsagObject(String serializedData) **throws** ParserException{

Agreement agreement = **null**;

**try**{

ObjectMapper mapper = **new** ObjectMapper();

agreement = mapper.readValue(serializedData, Agreement.**class**);

} **catch** (JsonProcessingException e) {

**throw** **new** ParserException(e);

} **catch** (Throwable e) {

**throw** **new** ParserException(e);

}

**return** agreement;

}

/\*

\* getWsagAsSerializedData receives in serializedData json

\* must return information following and xml in wsag standard.

\*/

@Override

**public** String getWsagAsSerializedData(String serializedData) **throws** ParserException {

StringWriter stringWriter = **new** StringWriter();

**try** {

Agreement agreement = getWsagObject(serializedData);

JAXBContext jaxbContext = JAXBContext.*newInstance*(Agreement.**class**);

Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

jaxbMarshaller.setEventHandler(**new** ValidationHandler());

jaxbMarshaller.marshal(agreement, stringWriter);

} **catch** (JAXBException e) {

**throw** **new** ParserException(e);

}

**return** stringWriter.toString();

}

/\*

\* getSerializedData receives in wsagSerialized the information in wsag standard as it was generated with the

\* getWsagAsSerializedData method and returns it in json

\*/

@Override

**public** String getSerializedData(String wsagSerialized) **throws** ParserException{

**try** {

JAXBContext jaxbContext = JAXBContext.*newInstance*(Agreement.**class**);

Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();

Agreement agreement = (Agreement)jaxbUnmarshaller.unmarshal(**new** StringReader(wsagSerialized));

ObjectMapper mapper = **new** ObjectMapper();

mapper.configure(SerializationFeature.*WRITE\_DATES\_AS\_TIMESTAMPS* , **false**);

String result = mapper.writeValueAsString(agreement);

**return** result;

} **catch** (JsonProcessingException e) {

**throw** **new** ParserException(e);

} **catch** (JAXBException e) {

**throw** **new** ParserException(e);

}

}

# Build the code

To build the source code:

* open a command window
* go to the root directory (the same level where you have the configuration.properties.sample file)
* type: **mvn -Dmaven.test.skip clean install**

The war file should be copied automatically to the tomcat.directory folder defined in the configuration.properties file.