# Table of Contents

1. Table of Contents ................................................................. i
2. Usage ................................................................................ 1
3. FAQ ..................................................................................
1 Usage

1.1 JlibSEDML How-to

1.1.1 Introduction

This document provides some help in getting started with using the jlibSEDML library 2.0.0, which provides API for creating and manipulating Level 1, version 1 SED-ML documents.

1.1.1.1 What can jlibsedml do for me?

jlibsedml is a pure Java library for creating, parsing, validating and manipulating SED-ML files and will run on Java 1.5 or later. Version 2.0.0 contains the following features:

1. Basic support for SED-ML
   - Up to date support for SED-ML level 1 version 1 final specification.
   - Inbuilt validation of documents against the SED-ML XML schema.
   - Semantic validation against the specification.
   - Full support for SED-ML specific MathML symbols sum, product, max and min.
   - Read/write support for SED-ML archive format.
   - Inbuilt support for ontologies (e.g., KiSAO) and model languages used in SED-ML.

2. Support for execution of SED-ML documents

SED-ML documents are designed to be executable by software to reproduce simulation outputs. jlibsedml provides support for this:

   - Support for manipulating and executing MathML sub-elements.
   - Inbuilt simulation execution framework. Clients need just hook in a simulation engine and define what simulation algorithms they support.
   - Full post-processing of raw result data.
   - Full support for manipulation of model files using XPath.

3. Support for developers

   - Javadoc for all public API methods.
   - Comprehensive test suite of over 400 JUnit tests.

The SED-ML Level 1, Version 1 specification is now fixed, so we aim for the API for this library to be stable, and that future releases supporting SEDML level 1 will be binary backwards-compatible with this release.

Jlibsedml does not run simulations itself, nor plot the results of a time course, so its use is not intended for end-users. Instead, this library provides a toolkit to assist Java application developers in reading and writing SED-ML documents. In case of any discrepancy between the behaviour of the library and the specification, the specification will be correct - and we would also be grateful if we could be informed in order to rectify the situation.

However, there are some examples in the source code on our Sourceforge repository which can be used to drive simulations using the Biomodels web-service and Copasi or VCell libraries, if you have those libraries installed.

1.1.1.2 About the authors

The authors of this software are Richard Adams, Ion Moraru and Anuradha Lakshminarayana.
1.1.1.3 Licensing

JlibSEDML is available under the terms of the MIT license:

Copyright (c) 2011 Richard Adams, Ion Moraru, Dan Vasilescu, Anuradha Lakshminarayana.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.1.2 Installation

This library requires Java 1.5 or later.

The zip archive contains the following resources:

- A jar of the jlibsedml library.
- A lib folder that contains libraries required by jlibsedml.
- A doc folder with Javadoc.
- A src folder with the Java source code of this release. Development code can be obtained from the project's Sourceforge repository, which can be checked out as a standard Eclipse Java project.
- This document and a README file.

All the jars in the lib folder need to be added to the runtime classpath of the application that is using jlibsedml.

Alternatively, if you use Maven, you can add the following to your pom.xml file:

```xml
<dependency>
    <groupId>org.jlibsedml</groupId>
    <artifactId>jlibsedml</artifactId>
    <version>2.0.0</version>
</dependency>
```

1.1.3 Creating a SED-ML document

This use case is for clients that wish to provide the facility to their end-users of exporting a simulation configuration to SED-ML format. We'll use a simple example model throughout this tutorial to illustrate the usage of the library.

The well understood Lotka-Volterra predator-prey model is described by the following equations which describe the rates of change of prey (x) and predators (y) over time:
\[
\begin{align*}
\frac{dx}{dt} &= px - qxy \\
\frac{dy}{dt} &= rxy - sy
\end{align*}
\]

With initial values \( x = 105, y = 8 \) and parameter values \( p = 0.4, q = 0.04, r = 0.02 \) and \( s=2 \), the following time course is achieved:

If we plot predator vs prey, we see a stable cycle is achieved:
Now let's write a SED-ML file to describe this simulation and its visualization:

1.1.3.1 The basics
There are five elements in a SED-ML document:

- A description of the simulation algorithms used, time steps etc.,
- A description of the models and where they can be obtained.
- Tasks - which link a model with a simulation.
- DataGenerators - these describe how the raw time-series data should be transformed (if at all).
- Output - either charts, 3D surface plots, or data sets, and their component elements.

To start off we can create new SED-ML document object:

```java
SEDMLDocument doc = new SEDMLDocument();
SedML sedml = doc.getSedMLModel();
```

1.1.3.2 Creating a simulation element:
Currently, SED-ML just supports standard time courses. We create a simulation with the following arguments:

- a unique id (used for cross-referencing).
- an optional human-readable name (this can be null or empty).
- the start time point.
- output start point.
- output end point (also the endpoint of the simulation).
- The number of time points to output. Note that this is not the number of time steps to be used by the solver (which is often adaptive).
- An Algorithm object describing the simulation algorithm used.
Algorithms are classified using the KisaoID system. In this case, the results were generated using a Runge-Kutta 4th order solver:
Algorithm algo = new Algorithm("KISAO:0000032")

See the section on #kisao for more details.

Simulation sim = new UniformTimeCourse("id", "name", 0, 0, 10, 100, algo);
will configure a simulation to start at time=0, and run for 10 time units with 100 time points in the
output. The output will also be generated from time=0.

Now we can add the simulation description to the top-level container.

sedml.addSimulation(sim);

1.1.3.3 Creating a model element:

Now we need to describe the model, and need to provide the following information:

- a unique id (used for cross-referencing).
- an optional human-readable name (this can be null or empty).
- The XML language used to describe the model.
- A URI used to locate the model:

```java
Model model = new Model("modelid", "any name", SUPPORTED_LANGUAGE.SBML_GENERIC.getURN(), "model.xml");
```

sedml.addModel(model);

In this case, we will bundle the model as a local file with the simulation description, so we give a
local file name - but the URI could be a reference to an entry in "Biomodels":http://www.ebi.ac.uk/
biomodels, for example. The class SUPPORTED_LANGUAGE.java provides a useful set of constants
for the language attribute.

1.1.3.4 Creating a task element:

A Task describes the application of a simulation to a model, linking the simulation and the model by
cross references.

- a unique task id (used for cross-referencing).
- an optional human-readable name (this can be null or empty).
- a reference to the id of the model to be used.
- a reference to the id of the simulation to be used.

```
Task task1 = new Task("task1Id", "Run basic model", model.getId(), sim.getId());
```

The utility of such an element is perhaps not so obvious until we have created several model
definitions and simulation configurations; then, the ability to combine these in tasks will be
demonstrated.

1.1.3.5 Creating data generators:

Now we need to define what the data output is for each species in the model we're interested in.
DataGenerator elements contain MathML which describes any transformation needed of the raw
output. In our example, we just want the raw values. Any variable referenced in the MathML should
be listed in a ListOfVariables section, which can be used to identify a species in a model. So :

```
ASTNode root = Libsedml.parseFormulaString("x");
```

This line converts a C-style string representation of a Math statement into an ASTNode, which we
then set into the DataGenerator. In this case we are just using the variable name, so the MathML will
look like:
More about the manipulation of MathML can be found below.

Now we have to define where the 'x' comes from, and from which model. SED-ML uses XPath queries to identify parts of a model. The class SBMLSupport.java provides some utility functions to identify parts of SBML models that are likely to be required by SED-ML. We plan to add additional support for other modelling languages such as CellML, NeuroML etc., in future.

```java
SBMLSupport support = new SBMLSupport();
Variable var = new Variable("x", "x",task1.getId(), support.getXPathForSpecies("x"));
// now add this variable to the data generator:
dgx.addVariable(var);
// and add the data generator to the document:
sedml.addDataGenerator(dgx);
```

We would also need to create data generators for the species 'y' and for time, as well.

For example,

```java
Variable vartime = new Variable("time", "time",task1.getId(),VariableSymbol.TIME);
```

will add the implicit variable 'time' as a variable.

1.1.3.6 Generating the output

The first output we'll create is a standard time course plot of time on the x-axis and the species amounts on the y-axis. To do this we'll create a Plot2D, and then define the Curves on the plot:

```java
Plot2D plot1 = new Plot2D("basicPlot", "basic Plot");
Curve cv1 = new Curve("curve1ID", "",false,false,time.getId(),dgx.getId());
Curve cv2 = new Curve("curve2ID", "",false,false,time.getId(),dgy.getId());
plot1.addCurve(cv1);
plot1.addCurve(cv2);
sedml.addOutput(plot1);
```

The second output will reproduce the cycle plot:

```java
Plot2D plot2 = new Plot2D("limitCycle", "limitCycle");
Curve cv3 = new Curve("curve3ID", "",false,false,dgx.getId(),dgy.getId());
plot2.addCurve(cv3);
sedml.addOutput(plot2);
```
1.1.3.7 Annotating the document

SED-ML provides **Notes** and **Annotation** elements whose concepts are similar to the elements of the same name in SBML. Briefly, **Notes** should contain annotation in the form of XHTML that can be displayed to an end-user, whereas **Annotation** elements are for software tools to persist information for their own use, in XML format using their own namespace.

jlibsedml provides support for manipulating these elements. Notes and annotations can be added to any element in SED-ML.

To add a note:

```java
Model model; // or any object inheriting from SedBase
Element para = new Element("p");
para.setText("An example paragraph");
Notes note = new Notes(para);
model.addNote(note);

// To remove a note:
model.removeNote(note);
```

Jlibsedml uses **Jdom** to manipulate raw XML elements. JDOM is included in this distribution. For notes, the XHTML namespace is added automatically.

The API to add/remove annotations is similar to that for Notes. However, clients need to set the namespace of their element. E.g.,

```java
Element para = new Element("myElement");
para.setText("An example annotation content");
para.setNamespace(Namespace.getNamespace("http://my.app.ns/level1"));
Annotation ann = new Annotation(para);
model.addAnnotation(ann);
```

1.1.3.8 Writing the document

1. Basic export

Before writing the document we might want to check for errors:

```java
for (SedMLError err: doc.validate()){
    System.out.println(err.getMessage());
}
```

and we can export the document as follows:

```java
String xmlAsString = doc.writeDocumentToString(); or
File mySEDMLFile = new File("sedml.xml");
doc.writeDocument(mySEDMLFile);
```

1. Export as a SED-ML archive.

This export provides a self-contained archive of the SED-ML file and its models. It is basically a zipped archive containing a SED-ML file and one or more model files. The URIs of the model 'source' elements in the SED-ML model must refer to the model file names, in this case.
SED-MLDocument doc; // our SED-ML doc
File modelFile = new File("model.xml"); // our model file

IModelContent [] models = Arrays.asList(new IModelContent []{
    new FileModelContent(modelFile)});
byte [] sedx = Libsedml.writeSEDMLArchive(new ArchiveComponents(models, doc),"mysedmlfilename");

Once we have the byte [] we can write in a standard way. E.g.,

File myoutputfile = new File("myoutputfile.sedx");
FileOutputStream fos = new FileOutputStream(myoutputfile);
fos.write(miase);
fos.flush();
fos.close();

1.1.3.9 Conclusions
This section has given a brief introduction to the API for writing SED-ML documents.

1.1.4 Reading a SED-ML document.
This section describes the use-case of a simulation tool reading a SED-ML file and extracting the information it needs to run a simulation.

1.1.4.1 Reading in a SED-ML file:
You can read in a SED-ML document either from file or from a String.

    SED-MLDocument doc = Libsedml.readDocument(sedmlFile);
    //or
    SED-MLDocument doc = Libsedml.readDocumentFromString(xmlString);

You can now check the document for errors:

doc.validate();
if(doc.hasErrors()) {
    // ...
}
SedML sedml = doc.getSedMLModel();

1.1.5 Accessing information to run a simulation:
Interpreting a SED-ML document requires several steps. We will use the use-case of reproducing an Output. The steps involved are as follows:

1.1.5.1 Running the model
1 For a given output, identify the Tasks that need to be run:
   a Identify the data-generators involved.
   b Collect the variables used in each data generator.
   c For each variable, find out which task should be run.
2 Check if we can run the simulations in each task:
a Check the KISAO id to see if the application supports that sort of simulation.
b Check the model language to see if the application supports that model type.

3 Locate the Models.
a Identify the models referred to by URIs (rather than references to other models in the SED-ML document).
b Obtain the model as an XML document.
c Apply any changes described in Change elements.

4 Execute the simulation!

1.1.5.2 Processing the results
In general, a simulator will produce a time-series of raw values for each variable in the model. In order to apply the SED-ML description, several steps are needed:

1 Map between the columns of data generated by the simulator, to the data generator elements.
2 Map the data generator elements' variables to model identifiers.
3 Perform any post-processing of data based on the maths in the DataGenerator elements.
4 Generate a data set, where each column corresponds to a data generator element.
5 Now display this information in a Plot or save as a Report.

From the above description, there is a fair amount of manipulation of the SED-ML document required, and also many stages where failures can occur. For example, execution of SED-ML document may fail because of any or all of the following:

• Errors in the SED-ML document itself.
• Inability of the software to execute the simulation.
• Inability to handle the model language.
• Failure in time series execution.
• Failure in mapping between generated data columns and the model elements, via the data generator elements.

1.1.5.3 A SED-ML execution framework
In this release, we have implemented a prototype SED-ML execution framework to help with some of the boilerplate code needed to reproduce a simulation experiment described in a SED-ML document.

This code is experimental and can be found in the org.sedml.execution package.

1 Executing a simulation
The class AbstractSedmlExecutor.java performs most of the steps described in the section 'Running the model' above, and provides some hooks for subclasses to implement:

```java
protected abstract boolean supportsLanguage(String language);
protected abstract boolean canExecuteSimulation(final Simulation sim);
protected abstract IRawSedmlSimulationResults executeSimulation(String model,
    UniformTimeCourse simConfig);
```

The class IRawSedmlSimulationResults defines a basic interface to access simulation data. Clients of the library can adapt a particular time series output to this interface for processing. For example, in our Lotka-Volterra example, we would expect to get 3 columns of output: Time, x, and y.

The class also enables clients to log failures or warnings via calls to:

```java
protected final void addStatus(ExecutionStatusElement statusItem);
```
To query for success/failure of execution by calls to:

```java
public final boolean isExecuted();
```

Furthermore, clients can add their own classes to help resolve and identify models:

```java
protected final void addModelResolver(IModelResolver resolver);
```

By default, a `FileModelResolver` is supplied which can look for models which use either absolute file paths or file paths relative to the application’s concept of the current working directory.

### 1. Processing the results:

The class `SedMLResultsProcessor2` contains methods to process a raw simulation result. Using this class, one can obtain a `ProcessedSedMLSimulationResult` that has had any processing applied to it.

#### 1. An example:

```java
SedML sedml;
Output output; // an output from the the above SedML

// your class which extends AbstractSedmlExecutor
SedmlExecutionImpl impl = new SedmlExecutionImpl(sedml, output);
// we get a map that maps a task to a time-series result using that task.
Map<Task, IRawSedmlSimulationResults> rawResults = impl.runSimulations();
if(!impl.isExecuted()) {
    // show failure & exit
}

SedMLResultsProcessor2 pcsr = new SedMLResultsProcessor2(sedml, output);
pcsr.process(rawResults);
if(pcsr.isProcessed()) {
    IProcessedSedMLSimulationResults processedRes = pcsr.getProcessedResult();
    // display or save results depending on output type.
} else {
    ProcessReport pr = processedRes.getProcessReport();
    // show processing errors to users..
}
```

### 1. Other code snippets

#### 1. Analyzing the KISAO id: kisao

```java
// check algorithm - can we read it ?
String kisao=siml.getAlgorithm().getKisaoID();
KisaoTerm kt = KisaoOntology.getInstance().getTermById(kisao);
if(!kt.is_a(KisaoOntology.ALGORITHM<WITH_DETERMINISTIC>RULES)) {
    // we can only handle ODE based systems; handle failure here
}
```
1. A Simple test analysing the model type:

```java
// check model - can we handle it??
Model mRead;
String lang = mRead.getLanguage();
if(!lang.contains("sbml")) {
    // we can read sbml only, so fail here if need be
}
```

1. Reading SED-ML archive files: If we were reading from a SED-ML archive that contained the model, we can use the following API:

Models are encapsulated by IModelContent objects.

```java
ArchiveComponents ac = Libsedml.readSEDMLArchive(
    new FileInputStream("mysedml.sedx"));
List<IModelContent> models = ac.getModelFiles();
IModelContent model1 = models.get(0);
```

The getName() method in IModelContent should return the model file name that is referenced in the SED-ML document:

```java
String modelXML = model1.getContents(); // returns an XML String of the model.
String modelFileName= model1.getName();
```

Getting the SED-ML document itself from the archive is straightforward:

```java
SEDMLDocument doc = ac.getSedmlDocument();
```

1.1.5.4 Manipulating MathML expressions mathml

There are two elements in the SED-ML specification that use mathematical expressions - the computeChange sub-element of Model which describes how a value in a model should be changed; and the DataGenerator element. In a SED-ML file, these are described using MathML, but in the object model, math expressions are represented as ASTNodes. For writing MathML expressions in SED-ML, these can be generated as follows:

```java
ASTNode root = Libsedml.parseFormulaString("sin(x)"));
```

or programmatically using methods in ASTNode.

Mathematical manipulation is provided by the JMathML library, which contains further information about its usage.

1.1.5.5 Vector manipulations

The SED-ML specification describes several MathML constructs which will apply to a range of values, not a single scalar. These are min, max, sum and product functions. These functions will apply on all the values obtained from a time series. Because SED-ML’s definition of these elements is a little simpler than the MathML specification, we have implemented these functions as cysymbol elements. So, for example, the normalized value of a variable \( x \):

\[
\frac{x - \text{min}(x)}{\text{max}(x) - \text{min}(x)}
\]

would appear as below in MathML:
JLibSEDML provides a mechanism to generate this XML from plain text: First of all we need to register a symbol factory with jMathML which will produce these csymbol objects. This is a one-off statement that be applied at application startup, for example:

SymbolRegistry.getInstance().addSymbolFactory(new SedMLSymbolFactory());

Now you are ready to parse from plain text:

ASTNode node = Libsedml.parseFormulaString("(x - min(x)) / (max(x) - min(x))");

I.e., these functions can be written as function calls. The only restriction is that only a single identifier is used as an argument to a vector function.

1.1.6 Current issues.

1.1.6.1 Synchronization and thread safety.
This library is not thread safe; it is up to clients to provide external synchronization if need be.

1.1.6.2 Object identity.
Objects with an id attribute should be unique within their enclosing element in a SED-ML document, and this attribute is immutable once an object has been constructed. Therefore, the id attribute is used
for the equals() and hashCode() methods. This means that adding two elements with the same value of their id attributes will
For ASTNodes, object identity is used as the basis for equals()/hashCode().

1.1.7 More information

1.1.7.1 Validation of SED-ML documents
This library provides a number of validation services to warn of any problems in the document, and can be found in the org.sedml.validation package. These include the following:

1. Validation against the SED-ML XML schema. This will catch basic XML syntax errors, and unknown or missing elements.
2. Cross reference validation. For example, if a task refers to a model ID that is not defined in this document, this will generate an error.
3. Uniqueness of identifiers. Values of 'id' attributes of identifiable elements in a SED-ML document need to be globally unique.
4. Time course validation - that the numerical values describing the time course are consistent. For example, outputStartTime must be \( \geq \) initialStartTime.
5. KISAO identifiers - a warning is generated if the value of the KisaOID attribute in an Algorithm element is not a valid KISAO identifier.
6. URI validator - checks if 'source' attributes in models can be converted into URI objects.
7. Cyclic dependencies in the model source references.

It's up to clients to decide how to handle these warnings. All validators are called by SEDMLDocument.validate(), or each validator can be called independently.

Clients can implement their own validators by implementing the interface ISedMLValidator.

1.1.7.2 More examples
More example SED-ML documents can be found in the examples folder of the SED-ML Sourceforge project.

1.1.7.3 Javadoc
The Javadoc for the library provides more detailed information on how to use the library.

1.1.7.4 Getting involved
If you would like to contribute code or bugfixes to this project we would be delighted to hear from you! Please suggest changes or new features in the Sourceforge tracker to begin with.