JSXM MAVEN PLUG IN

Computer Science Department

CITY College

An International Faculty of the

University of Sheffield

Konstantinos Margaritis

Last update: 05 October 2012

Version: v1.0

The following installation guide was tested with 3.7 Ingido Eclipse. and Juno 4 Eclipse.

Maven Installation

In order to build and run maven projects the maven plug in must be installed prior to any other operation.

1. In Eclipse(Indigo/Juno) : Go to Help and click install new software. Make sure you have the box Hide items that are already installed ticked.
   a. **Ingigo:**
      i. Enter url http://download.eclipse.org/releases/indigo and type in filter text the word maven.
   b. Juno:
      i. Enter url http://download.eclipse.org/releases/juno and type in filter text the word maven.
2. In case maven is not installed in your eclipse application you should see the 2 results for 2 different sections. Collaboration and general purpose tools. Expand the General Purpose Tools and click the m2e- maven Integration for Eclipse.
3. Click 2 times Next and accept the license agreement. Finally click Finish.
4. You will be prompted with window asking you to either restart Eclipse or Apply Changes now. Click restart now and wait until your Eclipse resumes.
5. To check that your installation was successful you can go to About Eclipse → Installation details→ Installed software tab. Check if m2e - Maven integration in Eclipse is present.

XML Validator Installation (Optional)

The xml validator is an eclipse plug in that will help you validate your xml and xsd files. It should be noted that the plug in is not needed for the internal validation that takes place in the jsxm maven plug in. Even if you do not have installed the xml validator any possible error(s) will display when you will build your maven project.

1. Eclipse Indigo and Juno are not bundled with the Web Tools Platform plug in. There is a slight difference between the 2 installations, which will be explained in the following steps.
2. Install Repositories:
   a. Eclipse Indigo
      i. Go to Help→Install new software and enter the url http://download.eclipse.org/webtools/repository/indigo/
   b. Eclipse Juno:
      i. Go to Help→Install new software and use the url http://download.eclipse.org/releases/juno which is already pre-populated in the list of software repositories.
3. Click Web Tools Platform (WTP) (latest version available) .
4. Click 2 times Next and accept the license agreement. Finally click Finish. You will be prompted with window asking you to either restart Eclipse or Apply Changes now. Click restart now and wait until your Eclipse resumes.
5. You can check the validator by right clicking on an xml or xsd file. The right click menu will offer the option Validate.

Downloading examples

All the examples are available at http://www.jsxm.org/downloads.html under the Maven Examples section. By expanding the list you can see all the available examples and you can click an example to download. All examples are compressed maven projects.

1. Download and extract the desired example.
2. Within Eclipse Click File→Import
3. On the import wizard Select General→Existing Projects into Workspace
4. Click Browse and navigate to the extracted folder of the desired example.
5. Select the folder and Click Open. Then click Finish.
6. Finally, the example you downloaded should appear in the Package Explorer as a Maven Project (The project icon will contain the letter M on the left corner).

Deploying examples

Before deploying the examples: if any of the examples have an error indicator right click on the example project →Maven→ Update Project Configuration.

- *Note for Juno: Update Project..*

1. Right click on any of the examples that you have downloaded Run As→ Maven Build→ enter "clean install " in the goals section and click Run.
2. You should see the test generation process and the execution of the tests. In case you want to change the k you can enter in the pom.xml and in the configuration section you can add <kInput></kInput> and set you desired input. The default k is 2.
3. After the successful build you can expand the src/test/java directory inside the example you are building and you will see the generated tests along with the adapter. In case you want to run the tests manually you can right click on [specName]AdapterTest.java and click Run as→JUnit test. You should see all the tests running successfully.
4. Note* Examples eShop and Cart do not have implementation thus there are not Junit tests generated.

POM.xml Requirements

1. The following properties must be present in order for the success deployment of a maven project with jsxm specifications.

```
<properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<pluginRepositories>
        <pluginRepository>
                <releases>
                        <enabled>true</enabled>
                        <updatePolicy>always</updatePolicy>
                        <checksumPolicy>fail</checksumPolicy>
                </releases>
                <id>jsxm</id>
                <name>JSXM Model Based Testing</name>
                <url>http://www.jsxm.org/maven2/</url>
        </pluginRepository>
</pluginRepositories>

<dependencies>
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.10</version>
                <scope>test</scope>
        </dependency>
</dependencies>

<build>
        <plugins>
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-surefire-plugin</artifactId>
                        <version>2.12</version>
                        <configuration>
                                <parallel>methods</parallel>
                                <useUnlimitedThreads>true</useUnlimitedThreads>
                        </configuration>
                </plugin>
                <plugin>
                        <groupId>org.jsxm.maven.plugin</groupId>
                        <artifactId>jsxm-maven-plugin</artifactId>
                        <version>1.0</version>
                        <configuration>
                        </configuration>
                        <executions>
                                <execution>
                                        <id>generateJSXM</id>
                                        <phase>generate-test-resources</phase>
                                        <goals>
                                                <goal>generate-jsxm</goal>
                                        </goals>
                                </execution>
                        </executions>
                </plugin>
```

```
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <version>2.3.2</version>
                                <configuration>
                                        <source>1.6</source>
                                        <target>1.6</target>
                                </configuration>
                        </plugin>
                </plugins>
  <pluginManagement>
         <plugins>
                <plugin>
                  <groupId>org.eclipse.m2e</groupId>
                  <artifactId>lifecycle-mapping</artifactId>
                  <version>1.0.0</version>
                        <configuration>
                          <lifecycleMappingMetadata>
                            <pluginExecutions>
                                <pluginExecution>
                                        <pluginExecutionFilter>
                                        <groupId>org.jsxm.maven.plugin</groupId>
                                        <artifactId>jsxm-maven-plugin</artifactId>
                                        <versionRange>[1.0,)</versionRange>
                                        <goals>
                                                <goal>generate-jsxm</goal>
                                        </goals>
                                        </pluginExecutionFilter>
                                        <action>
                                          <execute>
                                             <runOnIncremental>false</runOnIncremental>
                                          </execute>
                                        </action>
                                </pluginExecution>
                            </pluginExecutions>
                          </lifecycleMappingMetadata>
                        </configuration>
                </plugin>
          </plugins>
  </pluginManagement>
 </build>
```

Settings different goals

The goal included in the examples is generate-jsxm. Generate-jsxm is dependent to compile-jsxm and compile-jsxm is depedent on validate-xml. There is automatic resolution of all the required dependencies.
The goals that depend on other goals trigger the execution of the goals that they depend on and ensure the successful execution of the plug in. Available goals: The names of the goals are self-explainable.

JSXM Goals

```
<execution>
	<id>compileJSXM</id>
	<phase>compile</phase>
	<goals>
		<goal>compile-jsxm</goal>
	</goals>
</execution>
<execution>
	<id>generateJSXM</id>
	<phase>process-test-resources</phase>
	<goals>
		<goal>generate-jsxm</goal>
	</goals>
</execution>
<execution>
	<id>animateJSXM</id>
	<phase>compile</phase>
	<goals>
		<goal>animate-jsxm</goal>
	</goals>
</execution>
<execution>
	<id>banimateJSXM</id>
	<phase>compile</phase>
	<goals>
		<goal>banimate-jsxm</goal>
	</goals>
</execution>
```

Repast Goals

```
	<execution>
		<id>compileRepast</id>
		<phase>compile</phase>
		<goals>
			<goal>compile-repast</goal>
		</goals>
	</execution>
```

General Goals

```
	<execution>
		<id>validateXml</id>
		<phase>validate</phase>
		<goals>
			<goal>validate-xml</goal>
		</goals>
	</execution>
```

Note* Every goal that you add in the executions of the plug in it must be added in the goals section of the pluginExecutionFilter as goal.

Example: if you add in the executions this execution:

```
<execution>
	<id> compileJSXM </id>
	<phase>compile</phase>
	<goals>
		<goal>compile-jsxm</goal>
	</goals>
</execution>
```

You must also add the `<goal>compile-jsxm</goal>` in pluginExecutionFilter.

Goals Settings

Goal: compile-jsxm

<m2Home ></m2Home >

Sets the maven local repository. By default the value is the default value set by maven .

<tempTarget></tempTarget>
Sets the temp directory that temporary files from jsxm are going to be saved. The default value is inside the target directory of a maven project.

<testDirectory ></testDirectory >
Sets the test directory of the maven project. The default value is the src/test/java of the maven project.

<javaDirectory ></javaDirectory >
Sets the implementation directory of the maven project. The default value is the src/main/java of the maven project.

<jsxmSpecRootDir ></jsxmSpecRootDir >
Sets the directory for all the jsxm specifications. The default value is the spec directory under src → src/spec.

<mode ></mode >
Sets the mode of the maven project. Verbose and overwrite are the available options. By default the mode is set to overwrite.

<kInput ></kInput >
Sets the k value that is going to be used in the generation of tests. The default value is 2.

<jsxmSxmImports></jsxmSxmImports >
JSXM imports for the SXM.java. The default imports are
import org.jsxm.jsxmcore.core.*;\nimport org.jsxm.jsxmcore.types.*;\n

<jsxmSxmBaseImports ></jsxmSxmBaseImports >
JSXM imports for the SXM_base.java. The default imports are
import org.jsxm.jsxmcore.core.*;\n\n

<sxmBaseInheritance ></sxmBaseInheritance >
The type that the SXM_base.java is inheriting.
The default value is SXM

<kList></ kList >

The kList is a list of properties that allows the execution of examples with a different k than the default. In case an example contains more than one specification, then there is the possibility to need a different k value for some of the specifications but at the same time not changing the general k value. In order to use this tag the following configuration must be followed.

```
<kList>
        <property>
                <name></name>
                <value></value>
        </property>
</kList>
```

Consider the example Book_Borrower which contains 3 specifications. The specifications Book, Borrower and Library will use the k value 2 in case none configuration is specified. On the other hand, if you would like to execute the Library specification with a k=4, and Book with k=5 then you should consider the following.

```
<kList>
        <property>
                <name>Library</name>
                <value>4</value>
        </property>
        <property>
                <name>Book</name>
                <value>5</value>
        </property>
</kList>
```

The above setting will override the default k and will use the specified instead. The Borrower specification will use k=2. The name tag must be identical with the specification name.

Goal: compile-repast

<m2HomeRepast></m2HomeRepast >
Sets the maven local repository. By default the value is the default value set by maven .

<tempTargetRepast></tempTargetRepast>
Sets the temp directory that temporary files from jsxm are going to be saved. The default value is inside the target directory of a maven project.

<testDirectoryRepast ></testDirectoryRepast >
Sets the test directory of the maven project. The default value is the src/test/java of the maven project.

<javaDirectoryRepast ></javaDirectoryRepast >
Sets the implementation directory of the maven project. The default value is the src/main/java of the maven project.

<jsxmSpecRootDirRepast ></jsxmSpecRootDirRepast >
Sets the directory for all the repast specifications. The default value is the spec directory under src → src/spec.

<modeRepast ></modeRepast >
Sets the mode of the maven project. Verbose and overwrite are the available options. By default the mode is set to overwrite.

<kInputRepast></kInputRepast >
Sets the k value that is going to be used in the generation of tests. The default value is 2.

<repastCopyDirectory></ repastCopyDirectory >
Sets the directory which the [specName]SXM.java and [specName]SXM_base.java are going to be copied. There is no default value for this property. Omitting this property will cause build failure in the compile-repast goal.

<repastSxmImports></repastSxmImports >
Repast imports for the SXM.java. The default imports are
`import org.jsxm.jsxmcore.core.*;\nimport org.jsxm.jsxmcore.types.*;\nimport java.util.*;\nimport repast.simphony.engine.schedule.ScheduledMethod;\nimport repast.simphony.random.*;\nimport repast.simphony.query.space.grid.*;\nimport repast.simphony.space.grid.*;\nimport repast.simphony.space.*;\nimport repast.simphony.context.*;\nimport repast.simphony.util.*;\n"`

<repastSxmBaseImports ></repastSxmBaseImports >
JSXM imports for the SXM_base.java. The default imports are
import org.jsxm.jsxmcore.core.*;\nimport org.jsxm.spatial.SXMSpatial;\n

<sxmBaseInheritance></sxmBaseInheritance >
The type that the SXM_base.java is inheriting. The default value is
SXMSpatial

<kList></ kList >
The kList is a list of properties that allows the execution of examples with a different k than the default. In case an example contains more than one specification, then there is the possibility to need a different k value for some of the specifications but at the same time not changing the general k value. In order to use this tag the following configuration must be followed.

```
<kList>
        <property>
                <name></name>
                <value></value>
        </property>
</kList>
```

Consider the example Book_Borrower which contains 3 specifications. The specifications Book, Borrower and Library will use the k value 2 in case none configuration is specified. On the other hand, if you would like to execute the Library specification with a k=4, and Book with k=5 then you should consider the following.

```
<kList>
        <property>
                <name>Library</name>
                <value>4</value>
        </property>
        <property>
                <name>Book</name>
                <value>5</value>
        </property>
</kList>
```

The above setting will override the default k and will use the specified instead. The Borrower specification will use k=2. The name tag must be identical with the specification name.

* It should be noted that m2Home, tempTarget, testDirectory, javaDirectory and jsxmSpecRootDir should be left in their default values because changing the above values can cause build failures. You can change the values only if you are very familiar with maven and jsxm tool.

** SxmImports and SxmBaseImports are also vital for the execution of the tool and changing them can cause compilation errors, which will lead to build failures.

*** Configuration (parameters) can be used from different goals as long as they depend on the goal that provides the parameters. For example, all the configuration parameters available for compile-jsxm will be also provided to generate-jsxm due to the fact that generate-jsxm depends on compile-jsxm goal.

Goal: validate-xml

`<jsxmSpecRootDir ></jsxmSpecRootDir>`
Sets the directory for all the jsxm specifications. The default value is the spec directory under src → src/spec.

`<nameSpace></nameSpace>`
Sets the namespace used by the jsxm core and by maven-jsxm-plug in. The default namespace is **http://www.jsxm.org/schema.** The namespace must be the same as the namespaces used in the xsd schemas (specification, sets, definitions). The above namespace is used for the validation, creation and all the necessary operations needed regarding the xml files.

*jsxmSpecRootDir and namespace are vital for the execution and changes should be made with caution. Wrong namespace or spec directory will lead to build failures.

Goal: create-jsxm

`<jsxmList></jsxmList>`
The above list is used for entering the names that are going to be used in the creation of the folders, subfolders, specification.xml and sets.xml. There is no default value for the above parameter. The parameter is working as a list and every new name should be given through a new parameter. For example, if you want to create 3 specifications with names Test1, Test2, Test3 you should enter inside the jsxmList tag the tags below.

```
<param>Test1</param>
<param>Test2</param>
<param>Test3</param>
```

The above 3 tags will create 3 folders Test1, Test2, and Test3 under the spec directory with all the necessary subfolders needed by the maven-jsxm-plug in(specification, testAdapters and animation). Finally, the there will be created Test1.xml and Test1_sets.xml, Test2.xml and Test2_sets.xml, Test3.xml and Test3_sets.xml templates containing sample data. There is not limit in number of parameters. The above tag should contain at least one parameter in order to compile successfully.

*Note: There is the possibility to create subfolders and folders inside the subfolders. For example in order to create 2 folders Test1 and Test2 inside a Tests folder, the following params should be used

```
<param>Tests/Test1</param>
<param>Tests/Test2</param>.
```

`<jsxmRootDir></jsxmRootDir>`
Sets the directory for all the jsxm specifications. The default value is the spec directory under src → src/spec.

`<nameSpace></nameSpace>`
Sets the namespace used by the jsxm core and by maven-jsxm-plug in. The default namespace is **http://www.jsxm.org/schema.** The namespace must be the same as the namespaces used in the xsd schemas (specification, sets, definitions). The above namespace is used for the validation, creation and all the necessary operations needed regarding the xml files.

Goal: animate-jsxm

`<animateList></animateList>`
In case there is more than one specification in one example and you would like to specify which specification you want to animate then you should use the tag animateList and the tag param inside the animateList. There could be as many param as the specifications of a project. For example if you have 10 specifications in an example but you would like to animate only 4 of them, then the four names of the specifications must be used in the animateList.

```
<animateList>
        <param>Accont</param>
        <param>Accont2</param>
        <param>Automato</param>
        <param>Cart</param>
</animateList>
```

The above example specifes that Account, Account2, Automato and Cart will be animated.

`<animateAll></animateAll>`
A simple boolean variable for setting whether or not the tool will animate all the examples.

Goal: banimate-jsxm

`<banimateList></banimateList>`
The same functionality as animateList. but used for banimation.

`<banimateAll></banimateAll>`
The same functionality as animateAll but used for banimation.

General settings

1. For viewing the line numbers:
   Preferences→General→Editors→TextEditors→Show line numbers

2. For changing the limited console output.
   Preferences→Run/Debug→Console→Limit Console Output(uncheck)