

1 Loading AMIDST dependencies from a remote maven repository

Here we explain how to add the AMIDST dependencies in a maven project with Java 8 or higher. Alternatively, you might prefer following the video-tutorial in this link .

In this example, we will use a project containing only one class, though the procedure here explain could be used in any other maven project. You can check this link for getting more information about how to create a new maven project.

For using the AMIDST Toolbox, the **pom.xml** file will be modified. First, in the Project view (located on the left) select the file pom.xml of your project and open it:

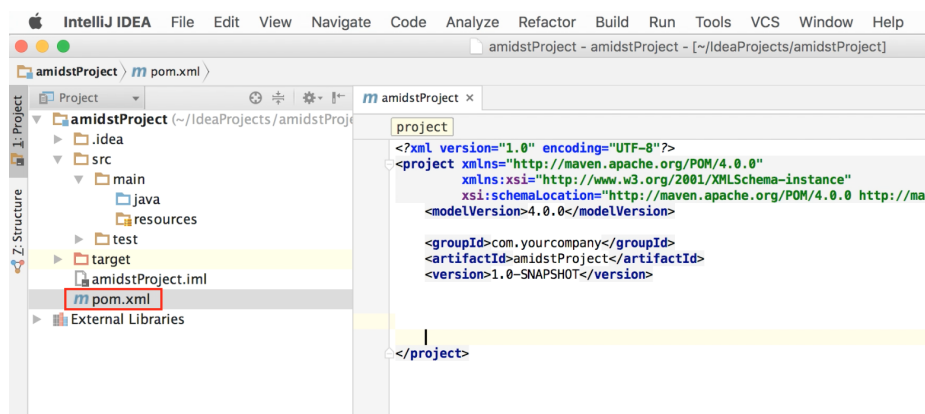


Figure 1: Capture of Maven project in IntelliJ and the initial pom.xml file

Add the AMIDST repository by including the following code to your pom:

```
<repositories>
<!-- AMIDST repository in github -->
<repository>
<id>amidstRepo</id> <!-- local identifier, it can be anything -->
<url>https://raw.github.com/amidst/toolbox/mvn-repo/</url>
</repository>
<!-- ... -->
</repositories>
```

Then, add the dependencies of modules in AMIDST you want to use. For each module, add an element **<dependency>...</dependency>** inside the labels **<dependencies></dependencies>**. For each one, we have to indicate the following information:

- **groupId** is an identifier of the project's module. In this case it should contain the value *"eu.amidst"*.
- **artifactId** is the name of the module we want to use. More precisely, it is the name of the jar file containing such module. You can see the list of AMIDST modules [here](#).
- **version** is the identifier of AMIDST Toolbox release. You can see [here](#) the list of all versions available.
- **scope** allows you to only include dependencies appropriate for the current stage of the build. We will set this to *"compile"*.

For example, for using the *core-dynamic* module, include the following code:

```
<dependencies>
<!-- Load any of the modules from AMIDST Toolbox -->
<dependency>
<groupId>eu.amidst</groupId>
<artifactId>core-dynamic</artifactId>
<version>0.4.2</version>
<scope>compile</scope>
</dependency>

<!-- ... -->
</dependencies>
```

Note that for using another module, simply change the value of the element `artifactId` (i.e. the content between the tags `<artifactId>` and `</artifactId>`). Now you can check in the **Maven Projects** panel that all the dependencies have been loaded:

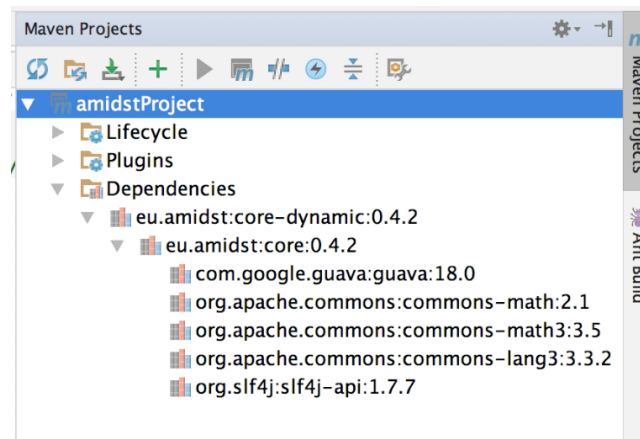


Figure 2: List of loaded dependencies

Note that the *core-dynamic module* depends on core that has been loaded as well. We recomend you to download the sources and the javadoc:

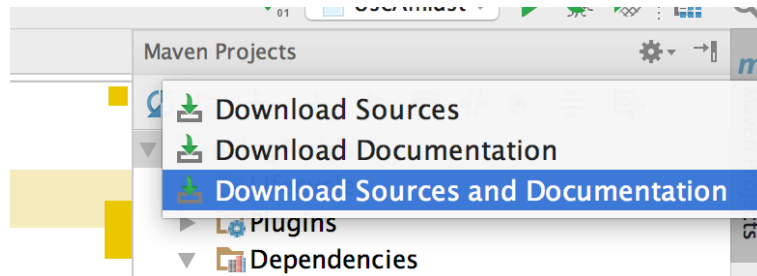


Figure 3: Download the JavaDoc and the source code

Finally, for testing purposes, we can run the code shown below that generates a random dynamic bayesian network (DBN) and prints its parameters.

```
import eu.amidst.dynamic.models.DynamicBayesianNetwork;
import eu.amidst.dynamic.utils.DynamicBayesianNetworkGenerator;

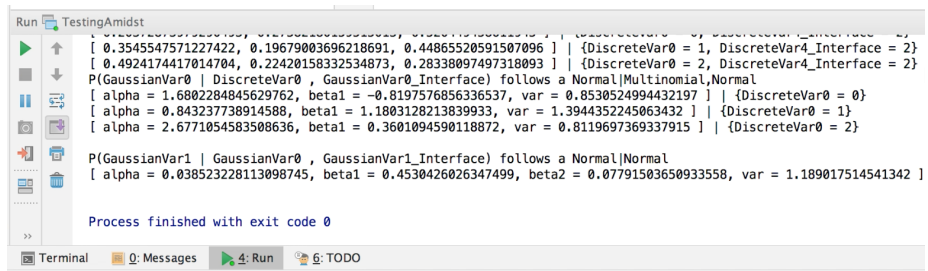
public class TestingAmidst {
    public static void main(String[] args) throws WrongConfigurationException {
        DynamicBayesianNetworkGenerator.setNumberOfContinuousVars(2);
        DynamicBayesianNetworkGenerator.setNumberOfDiscreteVars(5);
        DynamicBayesianNetworkGenerator.setNumberOfStates(3);

        DynamicBayesianNetwork extendedDBN =
            DynamicBayesianNetworkGenerator.generateDynamicBayesianNetwork();

        System.out.println(extendedDBN.toString());

    }
}
```

If everything goes right, the following output will be generated:



```
Run TestingAmidst
[ 0.3545547571227422, 0.19679003696218691, 0.44865520591507096 ] | {DiscreteVar0 = 1, DiscreteVar4_Interface = 2}
[ 0.4924174417014704, 0.22420158332534873, 0.28338097497318093 ] | {DiscreteVar0 = 2, DiscreteVar4_Interface = 2}
P(GaussianVar0 | DiscreteVar0, GaussianVar0_Interface) follows a Normal|Multinomial,Normal
[ alpha = 1.6802284845629762, beta1 = -0.8197576856336537, var = 0.8530524994432197 ] | {DiscreteVar0 = 0}
[ alpha = 0.843237738914588, beta1 = 1.1803128213839933, var = 1.3944352245063432 ] | {DiscreteVar0 = 1}
[ alpha = 2.6771054583508636, beta1 = 0.3601094590118872, var = 0.8119697369337915 ] | {DiscreteVar0 = 2}
P(GaussianVar1 | GaussianVar0, GaussianVar1_Interface) follows a Normal|Normal
[ alpha = 0.038523228113098745, beta1 = 0.4530426026347499, beta2 = 0.07791503650933558, var = 1.189017514541342 ]

Process finished with exit code 0
Terminal 0: Messages 4: Run 6: TODO
```

Figure 4: Ouput of the testing code that generates a random DBN