Ant tasks for AMMA

About Ant

Ant is a software tool for automating software build processes. It is similar to make but is written in the Java language, requires the Java platform, and is best suited to building Java projects.

The most immediately noticeable difference between Ant and make is that Ant uses XML to describe the build process and its dependencies, whereas make has its Makefile format. By default the XML file is named build.xml.

Ant is an Apache project. It is open source software, and is released under the Apache Software License.

The root tag of any ANT file is *project*. It has two optional attributes: *name* and *default*:

- *name* is just for convenience,
- *default* specify the default target to execute.

The syntax is as follow:

```
<project name="myProjectName" default="all">
      ...
</project>
```

Each project defines one or more *targets*. A target is a set of *tasks* you want to be executed. When starting Ant, you can select which target(s) you want to have executed. When no target is given, the project's default is used.

A target can depend on other targets. You might have a target for loading models and metamodels, for example, and a target for executing the transformation. You can only execute the transformation when you have loaded models first, so the transformation execution depends on the loading target. It is often advised to create multiple targets for better visibility in you Ant file.

A target has name and can depends on other tasks (referring by name to other tasks). The syntax is as follow:

```
<target name="loadModels">
      ...
</target>

<target name="transform" depends="loadModels">
      ...
</target>
```

A target can define *tasks*. A task is a piece of code that can be executed. A task can have multiple attributes (or arguments, if you prefer). The value of an attribute might contain references to a property. These references will be resolved before the task is executed.

Tasks have a common structure:

```
<name attribute1="value1" attribute2="value2" ... />
```

where name is the name of the task, attributeN is the attribute name, and valueN is the value for this attribute. Ant defines a set of built-in tasks (http://ant.apache.org/manual/coretasklist.html), along with a number of optional tasks (http://ant.apache.org/manual/optionaltasklist.html).

For more information on Ant itself, please consult its manual: http://ant.apache.org/manual/index.html.

AM3 Ant tasks

Task: am3.loadModel

This task is used to load a model either by modlel handler facilities or with injectors. This model may be a terminal model or a metamodel. The metametamodels are typically not loaded with this task since they come bundled with a model handler. These metametamodels are available under the special name strings `%EMF` for the Ecore metametamodel and `%MDR` for NetBeans/MDR MOF 1.4 metametamodel. MOF special name is contextually the metametamodel `%EMF` or `%MDR` depending on the model handler.
The am3.loadModel can have the following parameters:

Attribute	Description	Required	Default value
name	The name of the model in the Ant project.	Yes	None
metamodel	The name of the metamodel. This name must be equal to a previous model name loaded by am3.loadModel or to metametamodel special name `%EMF` or `%MDR`. If this name equals MOF, the modelHandler specific metametamodel is taken.	Yes	None
path	The path to the file of the model to load. It can be relative to the current directory (the one containing the Ant file). If absolute, the `'/'` root directory is the current workspace.	Yes	None
modelHandler	The model handler name to use for loading the model (`EMF` or `MDR`).	No	EMF

Loading of a metamodel with EMF (and Ecore as metametamodel):

```
<am3.loadModel modelHandler="EMF" name="News" metamodel="MOF"
path="metamodel/News.ecore" />
```

Equivalent to

```
<am3.loadModel modelHandler="EMF" name="News" metamodel="%EMF"
path="metamodels/News.ecore" />
```

Loading of a terminal model conforming to the previously loaded metamodel:

```
<am3.loadModel modelHandler="EMF" name="SampleNews" metamodel="News"
path="models/MyInput-News.xmi" />
```

By default, am3.loadModel is able to load file that the `modelHandler` can read. Sometimes, it is interesting to be able to load a model through an injector, for instance with the XML injector or reading a .km3 file and having its model conforming to KM3. This is possible with AM3 by using the nested parameter `injector`. For instance, if you want to inject an XML file to an XML model:

```
<!-- load XML metamodel -->
<am3.loadModel modelHandler="EMF" name="XML" metamodel="MOF"
path="metamodels/XML.ecore" />

<am3.loadModel name="myXML" metamodel="XML" path="inputs/MySample.xml">
  <injector name="xml" />
</am3.loadModel>
```

Task: am3.saveModel

This task is used to save a model using model handler facilities or with extractors. It is possible to save any model: terminal models, metamodels or metametamodels. The am3.saveModel can have the following parameters:

Attribute	Description	Required	Default value
name	The name of the model in the Ant project.	Yes	None
path	The path to the file of the model to save. It can be relative to the current directory (the one containing the Ant file). If absolute, the `'//'` root directory is the current workspace.	Yes	None

Saving of the previously loaded News metamodel:

```
<am3.saveModel model="News" path="outputs/NewsMM.ecore" />
```

You can see that the model attribute is the same as the `name` attribute of the previous am3.loadModel tasks. Once they are loaded, models are identified by this attribute name. Thus, you should avoid giving the same name for two different models. Each time it occurs, your previous model is overwritten.

By default, am3.saveModel is able to save model with `modelHandler` facilities (this model handler have been provided in the load model task). Sometimes, it is interesting to save a model with an extractor. For instance, if you have a model conforming to KM3, it can be interesting to use the KM3 extractor to save it as a .km3 file. Another example, if you have a model conforming to XML, you may want to get an XML document rather than the XML model. This can be done by specifying an `extractor` nested parameter. For instance, to extract a model conforming to the XML metamodel (in this case, the previously loaded XML model with XML injector):

```
<am3.saveModel model="myXML" path="outputs/SavingMySample.xml">
  <extractor name="xml"/>
</am3.saveModel>
```

Task: am3.atl

The purpose of this task is to execute an ATL transformation. The models used by a transformation are referenced by their name as defined at with the am3.loadModel task (name attribute).

An ATL task can have only one parameter specified as attributes: the path to the .atl file:

```
<am3.atl path="ATLFiles/MyTransformation.atl">
  ...
</am3.atl>
```

Within this task, you have to bind every model from the header of your ATL module. There is three kinds of nested parameters: `inModel`, `outModel` and `library`. The inModel kind is for source models; outModel for target models and library for helpers library.

For instance, if you have this module header:

```
module Families2Persons;
create OUT : Persons from IN : Families;
library myLib;
```

You have to create three inModel parameters (for `IN`, `Families` and `Persons`), one outModel (for OUT) and one library (for `myLib`). For instance, completing the previous sample of am3.atl task:

```
<am3.atl path="ATLFiles/MyTransformation.atl">
  <inmodel name="Families" model="..."/>
  <inmodel name="IN" model="..."/>
  <inmodel name="Persons" model="..."/>
  <outmodel name="OUT" model="..." metamodel="Persons"/>
  <library name="strings" path="lib/mylib.atl" />
</am3.atl>
```

Each parameter has a name that MUST be exactly the same as in the module header (case sensitive). For inModel parameters, `model` attribute refers to a name of a previously loaded model with am3.loadModel for instance. The attribute `model` of outModel do NOT refer a loaded model as it has not been yet created. The value of this attribute should be used latter as an identifier for the am3.saveModel task.

Every attributes for each nested parameters are summed here:

inModel:

Attribute	Description	Required	Default value
name	The name of the model in ATL module header.	Yes	None
model	The name of a model previously loaded	Yes	None

outModel:

Attribute	Description	Required	Default value
name	The name of the model in ATL module	Yes	None

	header.		
model	The name of a model previously loaded	Yes	None
metamodel	The name of the metamodel of the current model as it has been specified when loading	Yes	None

library:

Attribute	Description	Required	Default value
name	The name of the library in ATL module header.	Yes	None
path	The path to the ATL library file.	Yes	None

Launching an Ant file with AM3 tasks in an Eclipse workbench

Once you have defined your Ant file, right click on the file:

- Select Run As > Ant Build…
- Go to the JRE tab
- Select "*Run in the same JRE as the workspace*"