# Generating MPS models from Ant (obsolete)

## Generating MPS models from Ant



### Introduction

Users can generate their projects not only from MPS gui, but from Ant-scripts. For that MPS has a special Ant-task, `mps.generate`. The task classes are located in file %MPS_HOME%/lib/mps-backend.jar.

To use the task you should add a taskdef declaration into Ant-project:

```
<taskdef resource="jetbrains/mps/build/ant/antlib.xml"
        classpath="${mps.home}/lib/mps-backend.jar"/>
```

In the above code `mps.home` property points to base directory of MPS installation.

### Parameters

| Attribute | Description | Required | Default |
|---|---|---|---|
| compile | Indicates whether generated code requires compilation. | no | true |
| failonerror | Indicates whether generation errors will fail the build. | no | true |
| fork | Indicates whether to run generation in separate process. | no | true |
| loglevel | Controls the level of output messages shown. Should be one of "error", "warning", "info", "debug". | no | "info" |
| mpshome | Sets MPS location | Yes, if property mps.home is not specified in project. | |
| usepropertiesasmacro | Indicates that task should use properties defined in projects as MPS macro. | no | false |

### Nesteds

#### `modules, model`

Nested elements `modules` and `model` are filesets, specifying models and modules files to generate.

#### `project`

Nested element `project` is a fileset of project files to generate.

### jvmargs

Nested element `jvmargs` is used to pass command-line arguments to java virtual machine. This element can only be used unless `fork` attribute is set to false. Each command-line argument is specified via `arg` inner. `jvmargs` task can be used outside of any target, for example it can be specified in the beginning of a project and then be referenced in several tasks.

### library

Nested element `library` is used to define libraries, required for generation (in MPS ide they are located in "Settings">"Ide Settings">"MPS Library Manager").

| Attribute | Description | Required | Default |
|---|---|---|---|
| name | Library name. | yes | |
| dir | Library base directory. | yes | |

### macro

Nested element `macro` is used to specify values of path variables (in MPS ide they are located in "Settings">"Ide Settings">"Path Variables").

| Attribute | Description | Required |
|---|---|---|
| name | Path variable name. | yes |
| path | Path variable value. | yes |

## Configuration and caches directories

Places where MPS keeps its configuration and caches can be changed via `jvmargs` nested element. To do so, add the following children to `jvmargs`:

```
<jvmargs>
   <arg value="-Didea.config.path=%CONFIGURATION%"/>
   <arg value="-Didea.system.path=%SYSTEM%"/>
</jvmargs>
```

Replace `%CONFIGURATION%` and `%SYSTEM%` with path to the desired configuration and system directories. Note that ant task should be started in fork mode (`fork` property set to `true`) for using `jvmargs` element.

## JVM options

By default, since 2.0 Milestone 6, MPS tasks are started in fork mode, to be able to use more memory for generation. Default options provided to java machine are "-Xss1024k -Xmx512m -XX:MaxPermSize=92m -XX:+HeapDumpOnOutOfMemoryError -client". All of them, except for "-client", you can change via jvmargs element. If you experience `java.lang.OutOfMemoryError`, increase heap size using `-Xmx` argument, like it is set to 1024m below:

```
<mps.generate>
   <jvmargs id="myargs">
      <arg value="-Xmx1024m"/>
   </jvmargs>
   ...
</mps.generate>
```

## Examples

Generate project MyProject.

```
<mps.generate>
   <project file="MyProject.mpr"/>
</mps.generate>
```

Generate all models found in directory project1/languages/language1/languageModels. Fail if an error occurs during generation. Show debug messages.

```
<mps.generate failonerror="true" loglevel="debug">
   <model dir="project1/languages/language1/languageModels"/>
</mps.generate>
```

Generate model project1/languages/language1/languageModels/editor.mps. Note that base directory of language1, owner of model editor.mps, should be listed as library.

```
<mps.generate>
   <library name="language1" dir="project1/languages/language1/"/>
   <model file="project1/languages/language1/languageModels/editor.mps"/>
</mps.generate>
```

Generate projects project1, all modules from project2/languages/ folder and all models from project3/languages/language3/languageModels/ in separate process with -Xmx parameter of jvm set to 512m. Before generation load library someLibrary from lib/library1.

```
<jvmargs id="myargs">
   <arg value="-Xmx512m"/>
</jvmargs>

<mps.generate fork="true">
   <jvmargs refid="myargs"/>
   <library name="somelibrary" dir="lib/library1"/>
   <project file="project1.mpr"/>
   <modules dir="project2/languages/"/>
   <model dir="project3/languages/language3/languageModels/"/>
</mps.generate>
```

# Testing your models on TeamCity

MPS offers Ant-tasks for testing MPS models on TeamCity. Executed, those tasks output specially formatted messages, signalling tests start, finish, failure.

There are two types of tests: generation tests and broken references tests.

## Generation Tests

For generation tests `mps.test.generation` task is used. It has the same attributes and nesteds as `mps.generate` task, but also adds few more.

| Attribute | Description | Required | Default |
| --- | --- | --- | --- |
| invoketests | Automatically invokes unit tests in generated models. | no | false |
| showdiff | Calculate difference between files on disk and generated files and fails test if there is one. | no | false |

When running `mps.test.generation` task, each model is considered as single thing to test. That means, if models `model1` and `model2` are loaded, task will run two tests named "generating model1" and "generating model2". When nested `project` is used, task will load a project and read its "test configuration" to find out which models to generate. If no test configuration can be found for project, task will print a warning message. To override this behaviour one need to set attribute `wholeproject` to `true` in `project` inner. Then all models from project will be loaded.

## Difference between **`mps.test.generation`** and **`mps.generate`**

`mps.generate` does generation and compilation (if the corresponding option is set) of specified models, modules and projects. All used libraries have to be compiled before generation starts, so you have to set `compile` attribute to true for all libraries, distributed in sources. `mps.generate` always ensure that dependencies of a module are generated before module generation (if they are listed among targets to generate). `mps.test.generation` does not do so. You have to ensure that everything you need is generated and compiled before the task starts. It also does not save anything (like generated sources or compiled classes) on the disk and does not load compiled classes. It does only testing.

## Broken References Tests

For testing broken references `mps.test.broken.references` task is used. This task's parameters are the same as in `mps.generate` task.

Previous Next