Using simulation command line and showing test results through Jenkins

On this page

> **Note**
> In these scenarios, all related items on this page to be used are in the TestSimulationCommandLine.zip sample.

Preparing a simulation project and Configs in MagicDraw

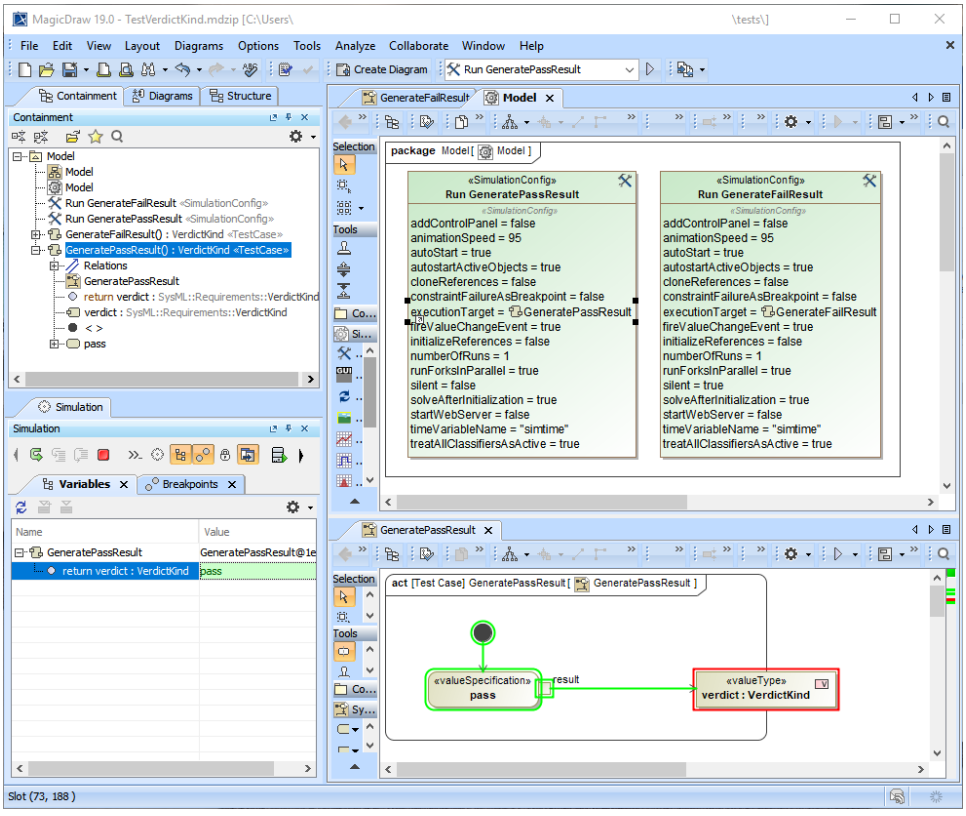
To prepare a simulation project and Configs in MagicDraw

---

1. Create a simulation model with Behaviors, e.g., *TestVerdictKind.mdzip*.
2. Apply «TestCase» to the Behaviors so the pass/fail status will be returned to the VerdictKind Activity parameters.
3. Create Simulation Configs and set those Behaviors as **executionTarget** of those Simulation Configs.

   > **Note**
   > All required resources must be available for the Simulation Configs, e.g., loadCSV and fmu. All features preventing simulation from the start /terminate, e.g., false autoStart, context without Classifier Behavior, chart windows at the end of execution, or output parameters at the end of execution (running Activity) will also be automatically started/terminated.

4. Run each Simulation Config to verify that an expected result, either pass or fail, must be returned.



Preparing a simulation project and Configs in MagicDraw.

Running a project through simulate command line

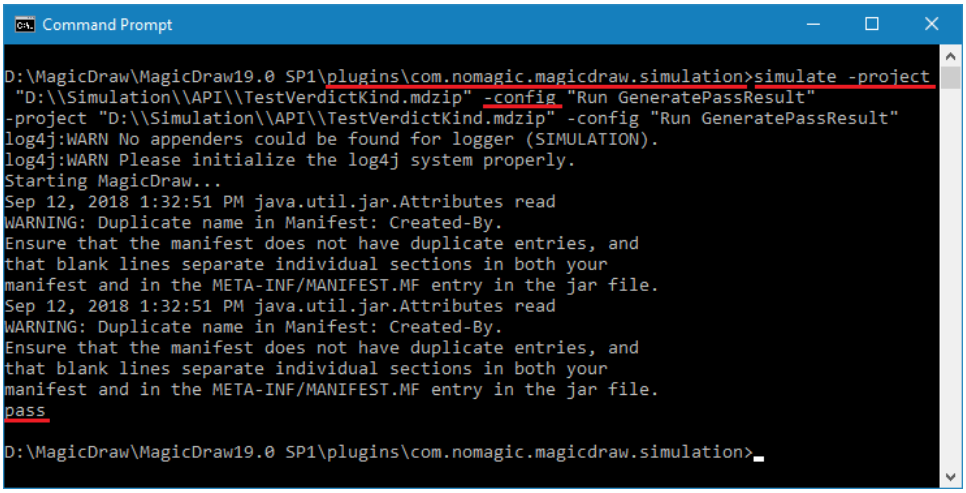
To run a project through simulate command line

1. Run the command line console, e.g., cmd on Windows.
2. In the console, type any commands to go to the local installation of MagicDraw.
3. Go to the **plugins\com.nomagic.magicdraw.simulation** folder.
4. Use the following **simulate** command syntax: *simulate -project "[Path of an MDZIP project]" -config "[Config]"*.

   In this case, the following commands are used:

   ```
   simulate -project "D:\\Simulation\\API\\TestVerdictKind.mdzip" -config "Run GeneratePassResult"
   simulate -project "D:\\Simulation\\API\\TestVerdictKind.mdzip" -config "Run GenerateFailResult"
   ```

5. At the last line of the command prompt, **pass** or **fail** appears. You can also see the details of execution through the Simulation Log File.



Running the project through the simulate command line.

Creating a JUnit test case and configuration file

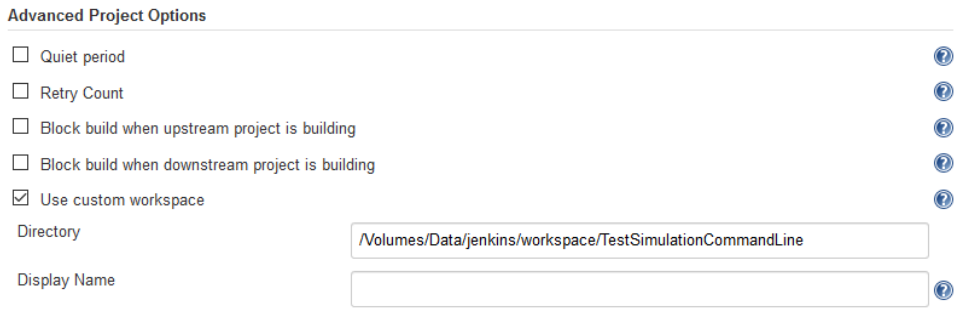
To create a JUnit test case and configuration file

1. Include the JUnit library in the Java project by creating a new Java project and adding the **JUnit 4** library in **current location** using the Eclipse's JUnit.
2. Create a new JUnit test case, e.g., Test Simulation Command Line.java.
3. Compile the test case and create a JAR file, e.g., **TestSimulationCommandLine.JAR** from the test case.
4. Create properties file, e.g., **TestGeneratePassResult.properties** and **TestGenerateFailResult.properties**.
5. Create an Ant configuration file, e.g., **run_junit.xml**.

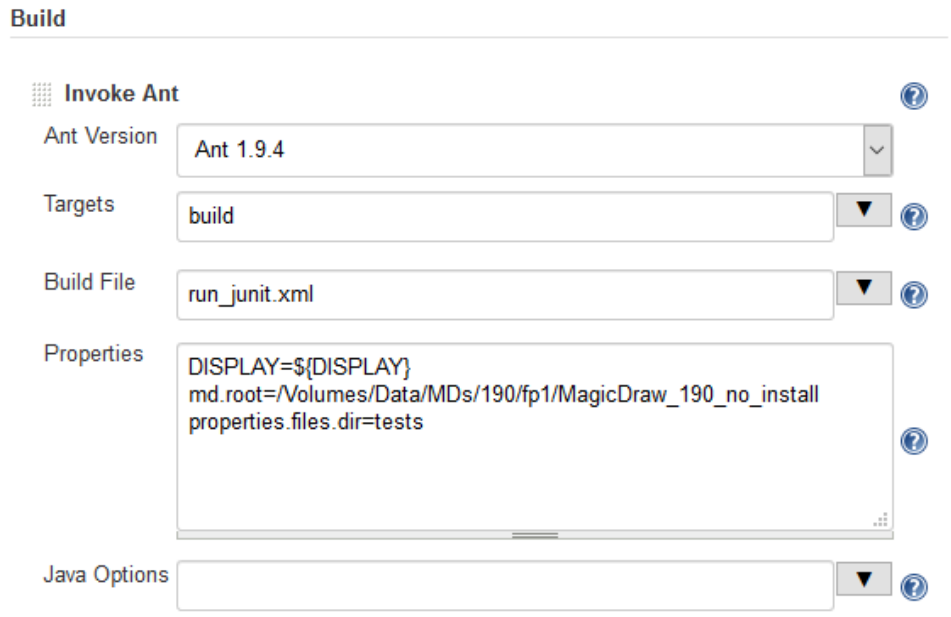
Configuring Jenkins for the Automated testing

To configure Jenkins for the Automated testing

1. Create a new Jenkins project.
2. Specify a **Project Name**, e.g., *Test Simulation Command Line*.
3. Go to **Advanced Project Options**, click **Advanced**, and select **Use custom workspace**.



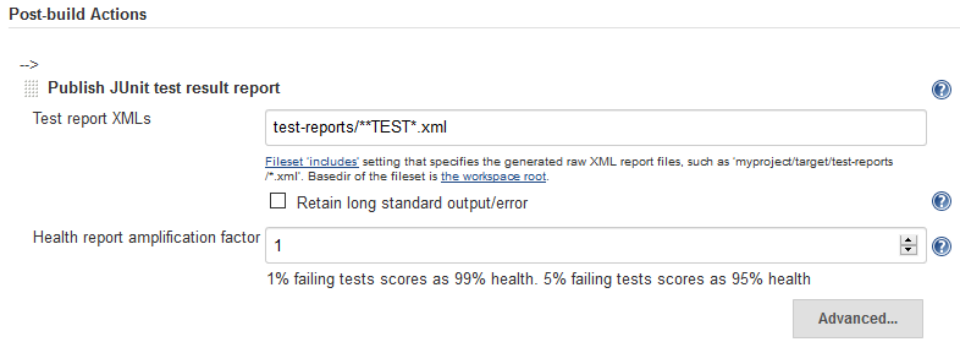
4. In the **Directory** box, specify a path to the directory that contains **run_junit.xml** (Ant configuration file).
5. Go to **Build** and click **Add build step**. Select **Invoke Ant**.
6. In the **Targets** box, type **build**.



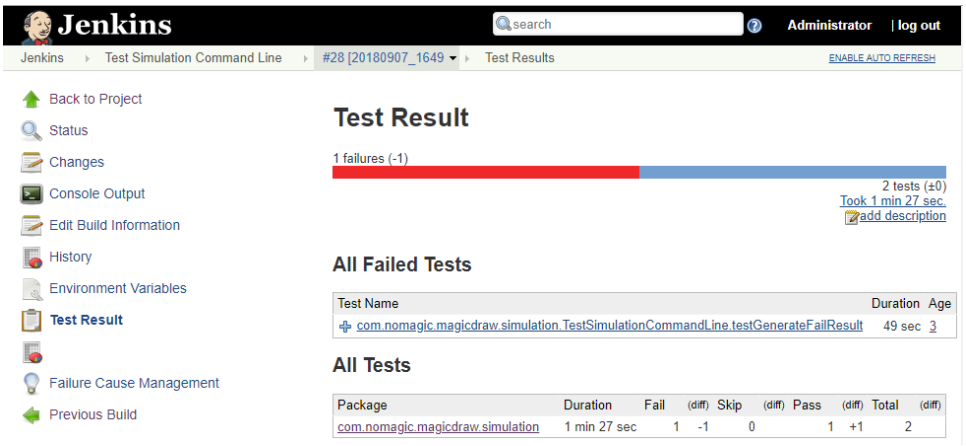
7. Select **Advanced**. In the **Build File** box, type **run_junit.xml**.
8. In the **Properties** box, specify *md.root=[MagicDraw installation directory]* and *properties.files.dir=[Path containing the properties files]*, as shown in the example above.

> **Tip**
> The path containing the property files can be either an absolute (full) path, e.g., *C:\Users\User\Desktop\TestSimulationCommandLine\tests* or relative path, e.g., *tests*.

9. Go to **Post-build Actions** and click **Add post-build action**. Select **Publish JUnit test result report**.
10. In the **Test report XMLs** box, specify a path to JUnit XML files, e.g., *test-reports/**TEST*.xml*.



11. Run the build. Test Result will be shown and allows you to view more details for those test cases.



Test Result of the simulation command line is shown on the Jenkins page.

Simulate command arguments and parameters

Examples of the **simulate** command lines are as follows:

- On Windows: *simulate.bat -project "../../samples/simulation/HingeMonteCarloAnalysis.mdzip" -config "Monte Carlo Analysis"*
- On Mac OS: *sh simulate.sh -project "../../samples/simulation/HingeMonteCarloAnalysis.mdzip" -config "Monte Carlo Analysis"*

Type of argument	Parameter	Description
Mandatory arguments	-project	Specify a file name with the path of a MagicDraw project. Both relative and absolute paths are supported.
	-config	Specify a SimulationConfig without a qualified name to run. **Note** If SimulationConfig has duplicate names (in different package), a warning message will be displayed in the Simulation log, and the first name will be used.
	-properties propertiesFileName	Specify the name of a properties file to use, e.g., in the TestSimulationCommandLine.zip sample. **Note** • If the **-properties** option is specified, the argument is the name of a properties file, and the mandatory argument is no longer required. • A properties file contains **-project** and **-config**, along with the specified data of each parameter.
Optional arguments	-inputs elementNameListWithValue	Specify one or more properties with values that will be provided to the Simulation Configuration.
	-inputsFile propertiesFileName	Specify the name of the properties file containing the parameters with values that will be provided to the Simulation Configuration. **Note** If you provide both **-inputs** and **-inputFile** arguments, parameters are collected from both arguments.
	-outputs elementNameList	Specify one or more properties whose values should be obtained during the simulation
	-outputsTemplate propertiesFileName	Specify the properties file that contains the parameters whose values should be obtained during the simulation. **Note** If you provide both **-outputs** and **-outputTemplate** arguments, parameters are collected from both arguments.
	-outputsFile propertiesFileName	Specify the properties file which will contain the output parameters with values after the simulation. The specified file will contain the parameters provided in the **-outputTemplate** or/and **-outputs** argument.
Optional (server) arguments	-server	Specify a name or an IP Address (and port) of the server.

	-leaveprojectopen	Leave the project open per property file after simulating the project, in a series of project simulation. The default value is **false**.
	-login	Specify a login name to log on to the server.
	-password	Specify a password to log on to the server.
	-spassword	Specify an encrypted password to log on to the server. **Note** To generate an encrypted password for this option, use the following command line (the characters of the encrypted password will be returned, e.g., *49034c0439....*): **generate -generatepassword -Administrator**
	-ssl	Specify to use and enable Secured Connection (SSL), **<true>** or **<false>** as the default value.
	-pversion	Specify the version of the server project.
	-tag	Specify the tag name of the server project. **Note** • **-tag** can't be used with **-pversion**. • If multiple projects are returned, only the latest version will be run.
	-readonly	Specify **true** (by default) to open the project as historic in the latest version without specifying **-pversion**. Specify other values to open the latest version normally.
Teamwork Cloud arguments	-branch	Specify a branch of Teamwork Cloud projects. **Note** • **-branch** and **-pversion** cannot be specified in the same command. • If **-pversion** and **-tag** are not specified, the latest version of the specified branch will be returned.
	-updatemodule	Specify update project usages if required, *<true>* or *<false>*, where *<false>* is the default.
	-projectpassword	Specify a project password.

Simulate command arguments with projects in Teamwork Cloud

To use simulate command arguments with projects in Teamwork Cloud, the projects must be added to Teamwork Cloud. The following examples demonstrate how to use the simulate command to run a model from Teamwork Cloud.

Scenario	Command argument
Simulate a project with an encrypted password and enable a secured connection (*-spassword, -ssl*).	*simulate -project "HingeMonteCarloAnalysis" -config "Monte Carlo Analysis" -servertype "twcloud" -server "localhost:1234" -login "Administrator" -spassword "49034c0439…" -ssl "true"*
Simulate a project by specifying a project version and project password (*-pversion, -projectpassword*).	*simulate -project "HingeMonteCarloAnalysis" -config "Monte Carlo Analysis" -servertype "twcloud" -server "localhost" -login "Administrator" -password "Administrator" -pversion "5" -projectpassword "Administrator"*
Simulate a project by specifying a tag name and branch (*-tag, -branch*). **Note** If there are any duplicated tags in the branch, the latest version will be run.	*simulate -project "HingeMonteCarloAnalysis" -config "Monte Carlo Analysis" -servertype "twcloud" -server "localhost" -login "Administrator" -spassword "49034c0439…" -tag "duplicatedTag" -branch "19.0 SP2"*
Simulate a command with three property files having different parameters. See Property file sample: Project-Config1.properties.	*simulate -properties "D:\\Simulation\\Project-Config1.properties" "D:\\Simulation\\Project-Config2.properties" "D:\\Simulation\\Project-Config3.properties"*
Simulate a project in Teamwork Server by specifying a project version (*-pversion*).	*simulate -project "HingeMonteCarloAnalysis" -config "Monte Carlo Analysis" -servertype "tw" -server "localhost" -login "Administrator" -password "Administrator" -pversion "2"*
Simulate a project in Teamwork Server by specifying an encrypted password and tag name (*-spassword, -tag*).	*simulate -project "HingeMonteCarloAnalysis" -config "Monte Carlo Analysis" -servertype "tw" -server "localhost" -login "Administrator" -spassword "49034c0439…" -tag "run2000"*

Simulate multiple projects in a single command with better performance (-*leaveprojectopen*). **Tip** When large-size projects need to be loaded and closed for many property files, it takes longer time to simulate projects. In this case, you can use **-leaveprojectopen** as an option to reduce memory for loading each project.	*simulate -leaveprojectopen true -properties "Properties1_Undefined.properties" "Properties2_Undefined.properties" "Properties3_Undefined.properties"*
Simulate a project in Teamwork Cloud while specifying several input parameters that will be provided for the Simulation Configuration (-inputs).	simulate -project "SpacecraftMassRollup" -config "spacecraft mass analysis" -inputs telecom.amplifier.me=8 telecom.antenna.me=29 -outputsFile “SimResults” -servertype "twcloud" -server "localhost” -login “Administrator" -spassword "*49034c0439…*"
Simulate a project in Teamwork Cloud while specifying the properties file that will be provided to the Simulation Configuration (-inputsFile), the properties file containing the parameters whose values should be obtained (-outputsTemplate), and the properties file that will contain output parameters with their values after the simulation (-outputsFile).	simulate -project "SpacecraftMassRollup" -config "spacecraft mass analysis" -inputsFile “D:\\Simulation\\InputsParams.properties” -outputsTemplate “D:\\Simulation\\OutputParams.properties” -outputsFile "D:\\Simulation\\SimResults.properties" -servertype "twcloud" -server "localhost” -login “Administrator" -spassword "*49034c0439…*"

Property file sample: Project-Config1.properties

```
project=Project1
config=Config1
servertype=twcloud
server=localhost
login=Administrator
spassword=49034c0439d3d1acc8d212adc289670c6224af46f2ec597eea1f25071740d5615a82d82d4460d5b95747f1e369ed26cdb5bb70fe6f50ff095cf978f1743
e5fbae6c4f2eef98abbca482133e4ca045c3ce407134e9c42966d2f245aed349d39fecd6a49f67b5019d5668e09cfd7f10a9363c5657a01addb9829052ffc26c49364
```

JUnit test sample: TestSimulationCommandLine.java

```
package com.nomagic.magicdraw.simulation;

import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Path;

import org.junit.Before;
import org.junit.Test;

/**
 * A test class for running the simulation command line and checking for the pass/fail result.
 * @author Chanon S.
 */

public class TestSimulationCommandLine {
        private static final String MD_ROOT_PROPERTY = "md.root";
        private static final String PROPERTIES_FILES_PROPERTY = "properties.files.dir";
        private static final String MD_DIR = System.getProperty(MD_ROOT_PROPERTY);
        private static final String SIMULATION_COMMAND_LINE = MD_DIR + "/plugins/com.nomagic.magicdraw.simulation/simulate.sh";
        private static final String PROPERTIES_FILES_DIR = System.getProperty(PROPERTIES_FILES_PROPERTY);
        private static final String PASS_PROPERTIES = "TestGeneratePassResult.properties";
        private static final String FAIL_PROPERTIES = "TestGenerateFailResult.properties";
        private static final String SYSTEM_NEW_LINE = System.getProperty("line.separator");

        @Before
        public void assertBuildExist() {
                System.out.println(MD_ROOT_PROPERTY + "=" + MD_DIR);
                System.out.println(PROPERTIES_FILES_PROPERTY + "=" + PROPERTIES_FILES_DIR);
```

```
                assertNotNull(MD_ROOT_PROPERTY + " is not specified.", MD_DIR);
                assertNotNull(PROPERTIES_FILES_PROPERTY + " is not specified.", PROPERTIES_FILES_DIR);

                assertTrue("MagicDraw directory does not exist, MD_DIR=" + MD_DIR, new File(MD_DIR).exists());
                assertTrue("Properties files directory does not exist,PROPERTIES_FILES_DIR=" +
PROPERTIES_FILES_DIR, new File(PROPERTIES_FILES_DIR).exists());
                assertTrue("Could not find simulate.sh file, SIMULATION_COMMAND_LINE=", new File
(SIMULATION_COMMAND_LINE).exists());
        }

        @Test(timeout = 120000)
        public void testGeneratePassResult() {
                // This project and Simulation Config generate "pass" as the result of TestCase.
                String propertyFilePath = PROPERTIES_FILES_DIR + "/" + PASS_PROPERTIES;
                String consoleOutput = runWithProperties(propertyFilePath);
                System.out.println("Console output=" + consoleOutput);
                boolean result = consoleOutput.contains(SYSTEM_NEW_LINE + "pass" + SYSTEM_NEW_LINE);
                assertTrue(consoleOutput, result);
        }

        @Test(timeout = 120000)
        public void testGenerateFailResult() {
                // This project and Simulation Config generate "fail" as the result of TestCase.
                String propertyFilePath = PROPERTIES_FILES_DIR + "/" + FAIL_PROPERTIES;
                String consoleOutput = runWithProperties(propertyFilePath);
                boolean result = consoleOutput.contains(SYSTEM_NEW_LINE + "pass" + SYSTEM_NEW_LINE);
                assertTrue(consoleOutput, result);
        }

        private String execute(String command) {
                System.out.println("Executing, command=" + command);

                StringBuilder result = new StringBuilder();
                try {
                        Process p = Runtime.getRuntime().exec(command);
                        BufferedReader input = new BufferedReader(new InputStreamReader(p.getInputStream()));

                        String line;
                        while ((line = input.readLine()) != null) {
                                result.append(line);
                                result.append(SYSTEM_NEW_LINE);
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return result.toString();
        }

        private String runWithProperties(String propFilePath) {
                String absPropPath = propFilePath;
                if (propFilePath != null) {
                        Path path = new File(propFilePath).toPath();
                        if (!Files.exists(path)) {
                                fail("Properties file <" + propFilePath + "> not found.");
                        } else {
                                absPropPath = path.toAbsolutePath().toString();
                        }
                }
                String cmd = propFilePath == null || propFilePath.isEmpty() ? SIMULATION_COMMAND_LINE :
SIMULATION_COMMAND_LINE + " -properties \"" + absPropPath + "\"";
        return execute(cmd);
        }
}
```

Ant configuration file sample: run_junit.xml

```
<project name="Run JUnit and Generate Reports" default="build" basedir=".">
        <target name="build" depends="set.properties, prepare, run.junit" />
        <target name="set.properties">
                <property name="md.root" location="${md.root}" />
                <property name="properties.files.dir" location="${properties.files.dir}" />
                <property name="test.reports.dir" location="test-reports"/>
        </target>
        <target name="prepare">
                <delete dir="${test.reports.dir}" />
                <mkdir dir="${test.reports.dir}" />
        </target>
        <target name="run.junit">
                <junit printsummary="yes" fork="yes">
                        <classpath>
                                <pathelement location="lib/TestSimulationCommandLine.jar"/>
                                <pathelement location="lib/junit-4.12.jar"/>
                                <pathelement location="lib/hamcrest-core-1.3.jar"/>
                        </classpath>
                        <jvmarg value="-Xmx1200m" />
                        <jvmarg value="-Xms256m" />
                        <jvmarg value="-XX:PermSize=128M" />
                        <jvmarg value="-XX:MaxPermSize=256M" />
                        <jvmarg value="-XX:-UseSplitVerifier" />
                        <jvmarg value="-noverify" />

                        <sysproperty key="md.root" value="${md.root}" />
                        <sysproperty key="properties.files.dir" value="${properties.files.dir}" />

                        <batchtest todir="${test.reports.dir}">
                                <zipfileset src="lib/TestSimulationCommandLine.jar">
                                        <include name="**/*.class"/>
                                </zipfileset>
                        </batchtest>
                        <formatter type="xml" />
                </junit>
        </target>
</project>
```