

USING THE OOSIML/JAVA

With a Terminal Window

On Linux Operating System

José M. Garrido
Department of Computer Science

Updated November 2016

College of Computing and Software Engineering
Kennesaw State University

© 2015, J. M. Garrido

1 The OOSimL Compiler

This document briefly describes the features and explains how to use the OOSimL compiler that generates Java code. The compiler that generates C++ code is explained in a separate document.

The OOSimL to Java compilation is a two-phase process. The first phase involves a translation into an equivalent Java source program, the second phase involves the corresponding Java translation to Java byte-code.

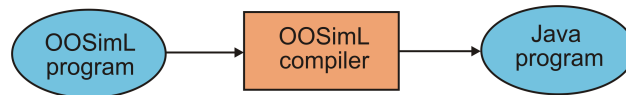


Figure 1: First phase of compilation of an OOSimL program

The first phase of the compilation is illustrated in Figure 1. The second phase of compilation involves compiling the corresponding Java program and use of the Java compiler. Figure 2 shows what is involved in compilation of a source program in Java. The Java compiler checks for syntax errors in the source program and then translates it into a program in bytecode, which is the program in an intermediate form.

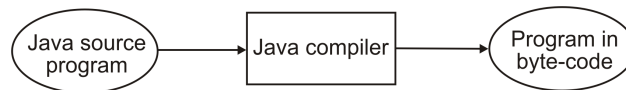


Figure 2: Compiling a Java source program

Note that the Java bytecode is not dependent on any particular platform or computer system. This makes the bytecode very portable from one machine to another.

Figure 3 shows how to execute a program in bytecode. The Java virtual machine (JVM) carries out the interpretation of the program in bytecode.

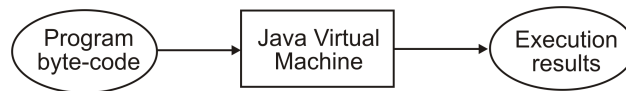


Figure 3: Executing a Java program

The OOSimL software, a sample model, and documentation are archived in the file `oosiml.tar.gz`, available from the OOSimL Web page:

`http://ksuweb.kennesaw.edu/~jgarrido/oosiml.html`

Currently, there are two general procedures used for developing programs and simulation models with the OOSimL compiler. These procedures are:

1. Developing the simulation model with the Eclipse integrated environment.
2. Working in a Terminal window.

The next sections introduce and explain how to configure and use the OOSimL compiler with the command window. The OOSimL software consists of two main components: the compiler/translator executable program `oosiml.out`, and the run-time library, `oosimlib.jar`. The compiler has the following functions:

1. Syntax checking of a source program in OOSimL; a list of errors is displayed and written to an error file with the same name as the source program
2. Generation of an equivalent program in Java

2 Installing the OOSimL Software and Models

The first general step is to create a folder and then to store the relevant files in the folder. As mentioned previously, the two essential files are: the OOSimL compiler, which is an executable program called `oosiml.out`, and the runtime library, `oosimlib.jar`.

1. Start and open a terminal window.
2. Create a new directory, for example `oosimldir`. Now change to that new directory.

```
mkdir oosimldir
cd oosimldir
```

3. Download the archive `oosiml.tar.gz` from the web page. Store the archive in the current directory and extract the files using the following command:

```
tar -xzf oosiml.tar.gz
```

4. Type the following command to uncompress the simulation model in the JAR file `carwash.jar`. This stores all the files of the Carwash model in a subdirectory `carwash`.

```
jar xf carwash.jar
```

5. To get a list of the files in the current directory, type `ls` and then press Enter.

3 Using a Terminal Window

Any text editor can be used to write and edit a new program in OOSimL or to modify an existing OOSimL program. A good editor for developing OOSimL and Java programs `gedit`. For editing and using the OOSimL compiler in a command window, complete the following steps:

1. Change the directory to the one where the files of the simulation model is stored. Use the `cd` command. For example, `cd carwash`.
2. Start the editor.
3. Type and edit the source program (e.g., `Carwash.osl`) with the editor. Note that the source files have the `.osl` extension.
4. Invoke the OOSimL compiler to check syntax and generate a Java program.

```
~/oosimldir/oosiml.out Carwash.osl
```

After a few seconds, you will see a message on the screen. In this case, File `Carwash.osl`, no syntax errors, lines processed: 49.

5. In the current directory, (or folder) two new files appear with the same name as your original source file. One of the files stores the syntax errors found. This file has the same name as the source program but with an `err` extension; for example, `Carwash.err`.

6. The second file contains the generated Java program; the file has the same name but with the `java` extension. For example, if your source program is `Carwash.osl`, then the Java program created is `Carwash.java`.
7. Use the Java compiler to compile the Java program (e.g. `Carwash.java`) generated by the OOSimL compiler. Compiling simulation programs, involve setting the *classpath* for the compiler to find the appropriate libraries, `oosimlib.jar`. Assume that these libraries are located in the `c:\oosimldirl` directory and that the current directory contains the Java classes for the simulation model, the following command compiles all the Java files in the current directory.

```
javac -classpath ~/oosimldirl/oosimlib.jar *.java
```

8. Use the Java Virtual Machine (JVM) to run the compiled program. When starting execution, enter the input data to the program, if needed. The following command will start of the simulation model, assuming that the main class is named *Carwash*.

```
java -classpath ~/oosimldir/oosimlib.jar;. Carwash
```

Note that running a simulation model creates two new files in the current directory, these are: `carwtrace.yxy` and `Carwstat.txt`. The names of these files can be changed by editing the source file with the main class.