Mesquite Software's CSIM for Java –
A Step-by-Step Explanation

By Herb Schwetman

Acknowledgements

The initial implementation of CSIM for Java was done by Conor Davis, based on Mesquite Software's CSIM 19 simulation toolkit for C and C++ developers.

Goals

CSIM for Java was written to allow Java programmers to quickly and easily code their discrete event simulations using the CSIM toolkit. Specifically, the goals for CSIM for Java are as follows:

- Provide all CSIM functionality using the Java programming language, with the same high standards of stability and quality that users expect from CSIM 19
- Feel natural to Java programmers
- Use standard Java constructs and tools
- Execute efficiently

This document gives a detailed explanation of the steps involved in the execution of a CSIM for Java model. It is intended to help the model builder, as he/she creates and debugs a simulation model. The explanation uses a sample program to illustrate the key points.

A Sample Program

```
// Generic application: Simple.java

import com.mesquite.csim.*;
import com.mesquite.csim.Process;
import com.mesquite.csim.file.Files;
import java.io.*;

public class Simple extends Model {
   public static void main(String args[]) {
      Simple model = new Simple();
      model.enableTrace(true);
      model.run();
   }
   public Simple() {
   super("Simple");
   }
   public void run() {
      start(new Sim());
   }

   private class Sim extends Process {
      public Sim() {
         super("Sim");
      }
      public void run() {
         add(new Gen());
         hold(2.0);
      }
   }

   private class Gen extends Process {
      public Gen() {
         super("Gen");
      }
      public void run() {
         while(true) {
            add(new Job());
            hold(rand.exponential(2.0));
         }
      }
   }

   private class Job extends Process {
      public Job() {
         super("Job");
      }
      public void run() {
         hold(rand.exponential(1.0));
      }
   }
}
```

Trace Output

```
      time      process  id   pri    status
 1    0.000          Sim   1     1    create Sim 1
 2    0.000          Sim   1     1    sched proc: t = 0.000, id = 2
 3    0.000          Sim   1     1    create Gen 2
 4    0.000          Sim   1     1    hold for 2.000
 5    0.000          Sim   1     1    sched proc: t = 2.000, id = 1
 6    0.000          Gen   2     1    sched proc: t = 0.000, id = 3
 7    0.000          Gen   2     1    create Job 3
 8    0.000          Gen   2     1    hold for 1.332
 9    0.000          Gen   2     1    sched proc: t = 1.332, id = 2
 10   0.000          Job   3     1    hold for 1.739
 11   0.000          Job   3     1    sched proc: t = 1.739, id = 3
 12   1.332          Gen   2     1    sched proc: t = 0.000, id = 4
 13   1.332          Gen   2     1    create Job 4
 14   1.332          Gen   2     1    hold for 2.351
 15   1.332          Gen   2     1    sched proc: t = 2.351, id = 2
 16   1.332          Job   4     1    hold for 0.626
 17   1.332          Job   4     1    sched proc: t = 0.626, id = 4
 18   1.739          Job   3     1    terminate process
 19   1.958          Job   4     1    terminate process
```

```
  20  2.000          Sim   1     1    terminate process
  21  2.000          Sim   1     1    halt simulation
Execution time: 0.11
```

Sequence of Methods, etc.

main() Line 9 – As with all Java applications, the *main()* method of *Simple* is the starting point. *Simple* extends the *Model* class (more on this later).

Line 10 – creates a new instance of *Simple*, called *Model.*

Line 11 – enables (turns on) trace mode; trace mode causes a file of CSIM actions to be written

Line 12 – calls the run method for the *Model* object (Line 17)

Line 17 – "starts" the model with the *Sim* process; this line first instantiates a *Sim* object (*Sim* extends *Process*, so this creates a new *Sim* process). After a new *Sim* is created, this process is "started." *Note: only the first process is started; all subsequent processes are "added".* The *model.start()* method initializes the runtime environment and then "adds" this instance of *Sim()* to the environment. Adding a process to the environment schedules the process to become "active" now (in simulated time). A process is assigned to a Java thread (from a pool of Java threads). The *start* method then activates the new process *Sim.run()* and waits for the model to "end".

Sim.run Line 26 – this instruction instantiates a new instance of the *Gen* process and then adds this instance to the environment (really, this instruction puts this instance of *Gen* on the "next event list," which is used to order the process "resumes" for all of the processes in the model (see below).

Line 27 – the *hold* statement refers to the *hold* method in the *Sim* Process object; the *hold(t)* method "schedules" this process to be "resumed" in ***t*** units of time in the future, which is the point in time defined by the current value of the simulation clock (*model.clock*) plus the value of t. The *hold* method then calls the *suspend_and_fire()* method; this instruction activates the *next* process and then suspends itself by executing the Java-thread-wait method.

Model maintains a list of waiting processes ordered by the time each process will be "resumed" – the next-event list. The *nextEvent* method removes the process at the head of this list, sets the clock to the time for this process activation, and the does *notifyAll* for this process.

Note: According the trace output, the value of the clock is still 0.000 (line 4), and the next-event list is as follows:
- *Gen, to resume at 0.000*
- *Sim, to resume at 2.000*

Gen.run Line 35 – the *run* method for the *Gen* process is the "code" for *Gen.* This code consists of a while loop that executes "forever" (really until the model ends). In every iteration of this *While* loop, *Gen* adds one new instance of the *Job* class (a *job* process – called *Job.3*) and then *holds* for an amount of time determined by the *rand.exponential(2.0)* function (Line 38). According to the trace, during the first iteration of the loop, the time for the *hold* is 1.332 units of time (line 8). The next-event list is as follows:

- *Job.3*, to resume at 0.000
- *Gen*, to resume at 1.322
- *Sim*, to resume at 2.000

Job.3.run Line 48 – The code for the *Job.3* process consists of one statement – *hold(rand.exponential(1.0)).* According to the trace output, the value for the *hold* statement is 1.739 – line 10. After the *hold* statement, the *Gen* process is suspended.

At this point, the value of the clock is 0.000 and there are three processes doing *hold* statements:

- *Gen*, to resume at 1.332
- *Job.3*, to resume at 1.739
- *Sim*, to resume at 2.000

Since *Gen* is the next process to resume, the clock will advance to 1.332 and *Gen* will be come active

Gen.run Line 37 – when *Gen* becomes active again (resumes), it "adds" another instance of the *Job* process (Job.4) – line 13;

Line 38 – *Gen* does another *hold()*; this time the value for the hold is 2.351 (line 14); the time it will be resumed is 1.332 + 2.351 = 3.683.

At this point, the value of the clock is 1.332 and there are three processes doing *hold* statements:

- *Job.4*, to resume at 1.332 (now)
- *Job.3*, to resume at 1.739
- *Sim*, to resume at 2.000
- *Gen*, to resume at 3.683

Job.4.run Line 48 – *Job.4* starts and executes the *hold* statement; this time, the value for the hold is 0.626. *Job.4* is scheduled to resume at time = 1.332 + 0.669 = 1.958, and suspends. The clock is 1.332, and the next event list is as follows:

- *Job.3*, to resume at 1.739
- *Job.4*, to resume at 1.958
- *Sim*, to resume at 2.000
- *Gen*, to resume at 3.683

Job.3.run Line 49 – The clock is now 1.739 (line 18); *Job.3* resumes, and the process terminates. *Note: A process automatically terminates when the* run *method ends.* When *Job.3* terminates, the process ends, and the next process on the next-event list will be resumed. The clock is 1.739, and the next-event list is as follows:

- *Job.4*, to resume at 1.958
- *Sim*, to resume at 2.000
- *Gen*, to resume at 3.683

Job.4.run Line 49 – The clock is now 1.958 (line 19); *Job.4* resumes, and the process terminates. When *Job.4* terminates, the process ends, and the next process on the next-event list will be resumed. The clock is 1.958, and the next-event list is as follows:

- *Sim*, to resume at 2.000
- *Gen*, to resume at 3.683

Sim.run Line 28 – *Sim* becomes the active process (line 20). When *Sim* ends (terminates), this is a special case (process id is 1), and the model halts (line 21). When the model halts, the *model.run* (Line 12) completes. In this example, there are no statements after the *model.run* statement, so the program completes.

If you have any further questions about CSIM for Java operation, please consult our online documentation at www.mesquite.com/documentation or contact us at Mesquite Software at info@mesquite.com or +1 512.338.9153.