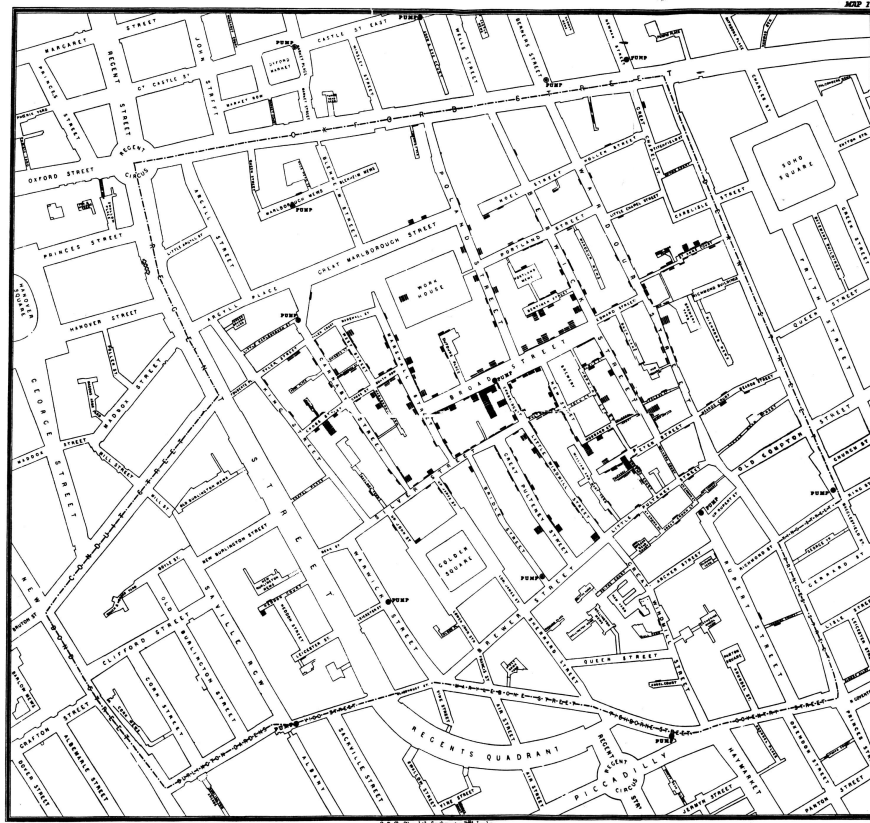

Broadwick

User Guide & Manual





Published by C.F. Cheffins, Lith, Southampton Buildings, London, England, 1854 in Snow, John. On the Mode of Communication of Cholera, 2nd Ed, John Churchill, New Burlington Street, London, England, 1855.

This image was originally from <http://en.wikipedia.org/wiki/File:Snow-cholera-map-1.jpg>

Cover Picture.

A variant of the original map drawn by Dr. John Snow (1813-1858), a British physician who is one of the founders of medical epidemiology, showing cases of cholera in the London epidemics of 1854, clustered around the locations of water pumps.

This image is in the [public domain](#) because its copyright has expired. This applies to Australia, the European Union and those countries with a copyright term of life of the author plus 70 years.

Table of Contents

COVER PICTURE.	2
INTRODUCTION	4
LICENSE	4
USING BROADWICK	4
CREATING A NEW PROJECT	4
CONFIGURATION FILES	8
EXTENDING THE MODEL	14
PACKAGES	15
ALGORITHMS	15
APPENDIX	16
MAVEN AS A BUILD TOOL	16

Introduction

In the event of an outbreak it is important to have modelling tools in place to estimate the likely origin, speed of spread, and size, and to be able to predict the impact of intervention measures. However, different diseases in different animal populations require somewhat different approaches due to the detection, transmission and recovery (or not) characteristics of hosts infected with the pathogen, and the contact patterns between susceptible hosts (whether of the same species or not). Consequently specific models developed for one disease system are unlikely to be entirely suitable for another.

Broadwick is a framework for developing sophisticated epidemiological based mathematical models, and consists of several Java libraries and bespoke packages. The components of Broadwick are written in such a way that a scientist may combine them in order to rapidly prototype a model for a new specific scenario.

- Supports single (e.g. within herd) or structured populations (e.g. multi-species or locations)

- Inclusion of movement over network data (e.g. Cattle movement Tracing System)

- Stochastic Individual Based simulations (including fast approximate options)

- Approximate Bayesian Computation inference for estimating model parameters from data via simulations

- Markov Chain Monte Carlo inference for estimating model parameters from data

License

Broadwick is released under the Apache 2 license.

Using Broadwick

Creating a New Project

Broadwick contains a set of packages that can be used as required. The framework is designed to be flexible and does not place any requirement on the user on how to use the framework. It is possible to use the classes and packages of Broadwick without using the powerful framework, creating your own main() method and taking responsibility for reading data files and configuration items though this is not the recommended way of using Broadwick.

Using The Command Line

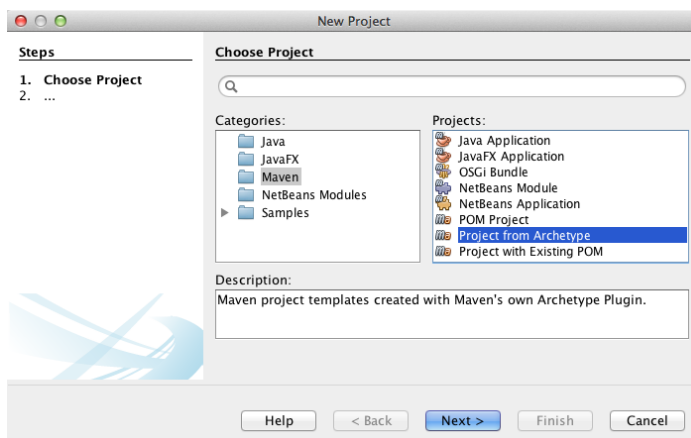
The Broadwick distribution contains a maven archetype for generating a skeleton project that contains all the configuration files, source code etc that is required to start a project based upon Broadwick. It uses apache maven as it's build tool. To generate a skeleton using this archetype on the command line (assuming that the broadwick-archetype jar is in your local repository)

```
mvn3 archetype:generate -DarchetypeGroupId=broadwick -DarchetypeArtifactId=broadwick-archetype \
-DarchetypeVersion=1.1 -DgroupId=broadwick.proj -DartifactId=StochasticSir -Dversion=0.1 \
-Dpackage=broadwick.stochasticsir
```

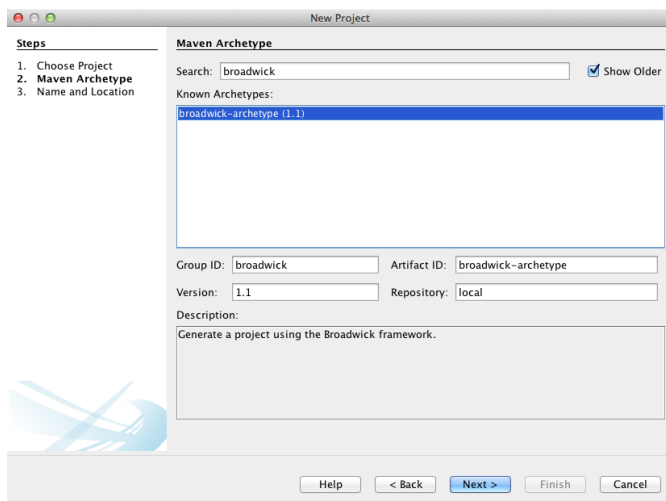
The groupId (maven uses the group id to uniquely identify your project), artifactId (is the name of your generated jar file without a version), version (the version number for your generated project) and the package to which the generated source will be created can be changed by modifying the -DgroupId, -DartifactId, -Dversion and -Dpackage arguments above.

Using Netbeans

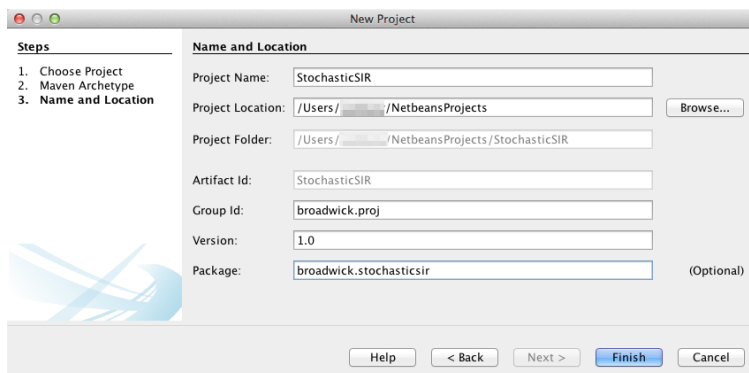
It is possibly easier to create a project using Netbeans (a free IDE available from Oracle, the 'owners' of Java). Open the Netbeans IDE and select File->New Project and choose a Maven project and "Project from Archetype" from the list of projects.



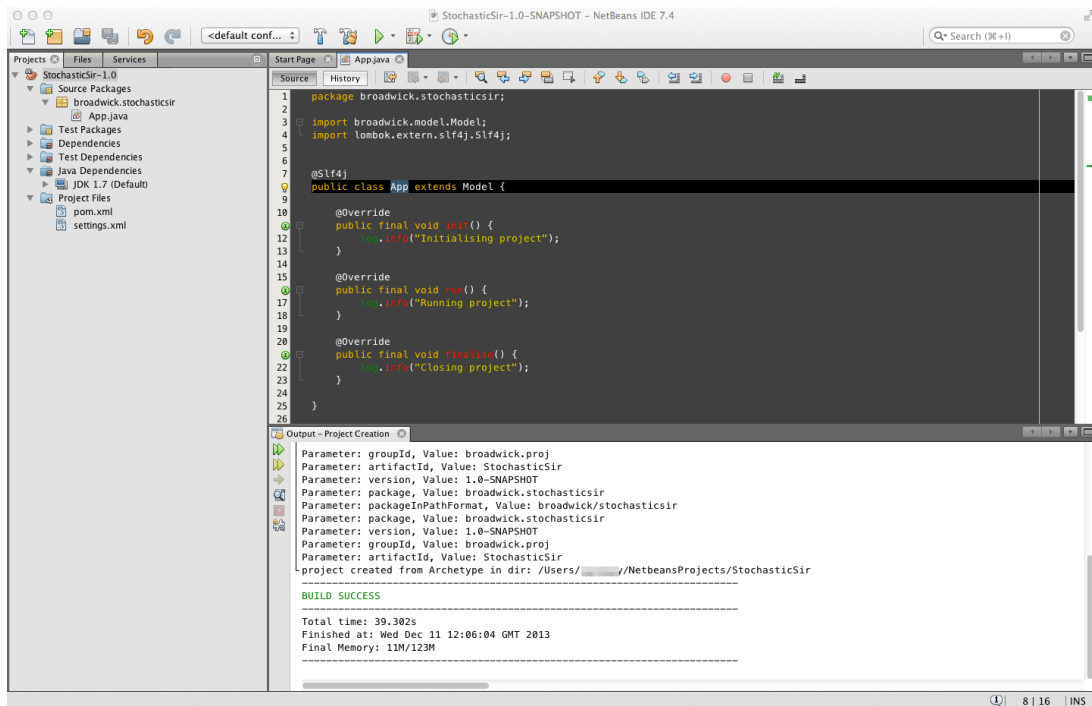
Click Next and choose the latest version of the broadwick-archetype from the "Known Archetypes". The version of the broadwick archetype corresponds to the version of Broadwick.



The projects details can be specified on the next screen.



Clicking “Finish” will create the project.




A number of minor changes are needed in the generated project.

Select the name of the class (“App”) and right-click and select Refactor->Rename; Change the name of the class to StochasticSIR.

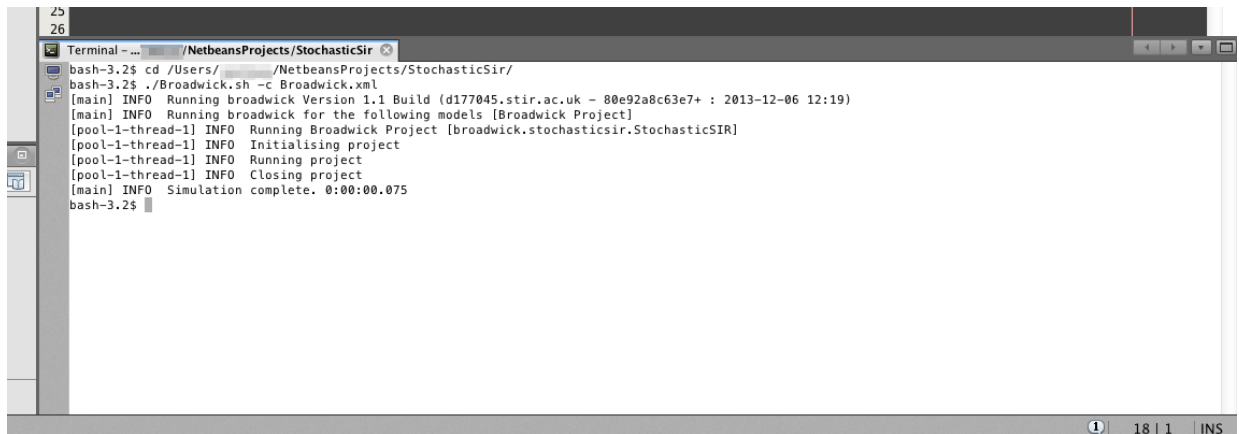
In the Broadwick.sh file change the \${artifactId} and \${version} to the artifactId and version specified when the project was created (StochasticSIR and 1.0 respectively). This is a shell script for running your project on Unix based systems, you will need to make it executable.

In Broadwick.xml (the configuration file for the generated project) change the name of the <classname> element to reflect the package and class (broadwick.stochasticsir.StochasticSIR)

We can build the generated project by selecting Run->Build Project from the menu bar or clicking the  icon in the toolbar. This will create a directory called target that contains, among other items, a jar file containing the compiled code and an executable jar file ending in .one-jar.jar. The Broadwick.sh file is a shell script that will run the executable jar file on *NIX systems.

The Broadwick.xml file contains the configuration for the project that we have just created. To run the newly compiled project using this configuration file open a terminal window (Netbeans contains a terminal window which can be accessed by selecting Window->IDE Tools->Terminal). Broadwick can be run using the following command

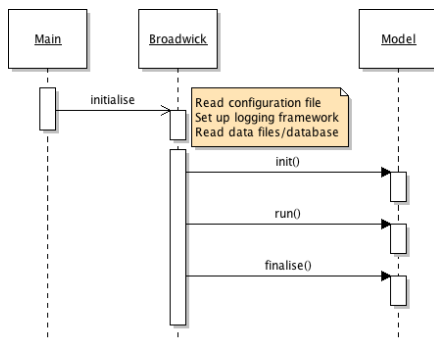
Broadwick.sh -c <configuration file>



```
25
26
Terminal - ... /NetbeansProjects/StochasticSir
bash-3.2$ cd /Users/.../NetbeansProjects/StochasticSir/
bash-3.2$ ./Broadwick.sh -c Broadwick.xml
[main] INFO Running broadwick Version 1.1 Build (d177045.stir.ac.uk - 80e92a8c63e7+ : 2013-12-06 12:19)
[main] INFO Running broadwick for the following models [Broadwick Project]
[pool-1-thread-1] INFO Running Broadwick Project [broadwick.stochastic.sir.StochasticSIR]
[pool-1-thread-1] INFO Initialising project
[pool-1-thread-1] INFO Running project
[pool-1-thread-1] INFO Closing project
[main] INFO Simulation complete. 0:00:00.075
bash-3.2$
```

When Broadwick starts it looks for all the models specified in the <model> elements in the projects configuration file. It creates objects for each <model> found using the default (empty) constructor for the class given in the <classname> element of the model (this is why no constructor is generated for the project).

For each project object created Broadwick will call the init(), run() and finalise() methods in turn. In our skeleton project we simply logged the fact that these methods were called. A simplified outline of how Broadwick initialises itself is shown below.



A description of the configuration file is outlined in the next section.

Configuration Files

The configuration file MUST conform to the Broadwick.xsd specification that is supplied with the Broadwick source code. The configuration is contained within the <project> </project> tags and contains the following tags

<logs>	<console>	Contains <level> and <pattern> tags to define the level and structure of logging messages displayed on the console.
	<files>	Contains <level> and <pattern> tags like the <console> log but also a <name> for the name of the file to contain the log messages and a boolean <overwrite> tag that lets Broadwick know if any existing file with that name should be overwritten.
<data>	<databases>	Contains the <name> tag to specify the location of the database.
	<datafiles>	Contains the <DirectedMovementFile> <FullMovementFile> <BatchMovementFile> <PopulationFile> <LocationsFile> and <TestsFile> tags for the various file structures recognised by Broadwick. See the following tables in this section for a description of each.
<models>	<model>	Broadwick can run several models concurrently (though they will share the same logging and data configurations).

Each model element contains the following

<classname>	Exactly one classname element that is the fully qualified name of the java class (that implements the broadwick.model.Model interface). This class MUST have a no argument constructor that Broadwick will use to create an instance through reflection.
<priors>	<p>The <priors> tag can contain as many elements as necessary, each prior contains an id attribute and optional <hint> and <initialVal> elements. The following priors are recognised by Broadwick</p> <p><uniformPrior> additionally contains <min> and <max> elements for uniformly distributed priors.</p> <p><gaussianprior> additionally contains <mean> and <deviation> tags for normally distributed priors.</p>
<parameter>	The parameters for the model can be encoded in this tag using the id and value attributes to name the parameters and give the parameter value. These can be accessed by the broadwick.model.Model.getParameterAs[TYPE](id) method

	that retrieves the parameter with the given id and converts it the the required type.
--	---

Each datafile contains specific information on it's layout, i.e. the columns in the file where the required data can be found. The structure of each recognised data file is outlined below.

DirectedMovementFile:

<name>	The name (including path from the configuration file) where the file can be found.
<alias>	An alias for the file.
<separator>	The character separating the columns in the datafile, e.g. “,” “\t”.
<idColumn>	
<movementDateColumn>	
<movementDirectionColumn>	
<locationColumn>	
<speciesColumn>	
<dateFormat>	
<customTags>	This optional field allows for optional information to be stored in the database.

FullMovementFile:

<name>	The name (including path from the configuration file) where the file can be found.
<alias>	An alias for the file.
<separator>	The character separating the columns in the datafile, e.g. “,” “\t”.
<idColumn>	
<departureDateColumn>	
<departureLocationIdColumn>	
<destinationDateColumn>	
<destinationLocationIdColumn>	
<marketIdColumn>	
<marketDateColumn>	
<speciesColumn>	
<dateFormat>	
<customTags>	This optional field allows for optional information to be stored in the database.

BatchMovementFile:

<name>	The name (including path from the configuration file) where the file can be found.
<alias>	An alias for the file.
<separator>	The character separating the columns in the datafile, e.g. “,” “\t”.
<batchSizeColumn>	
<departureDateColumn>	
<departureLocationIdColumn>	
<destinationDateColumn>	
<destinationLocationIdColumn>	
<marketIdColumn>	
<marketDateColumn>	
<speciesColumn>	
<dateFormat>	
<customTags>	This optional field allows for optional information to be stored in the database.

PopulationFile:

<name>	The name (including path from the configuration file) where the file can be found.
<alias>	An alias for the file.
<separator>	The character separating the columns in the datafile, e.g. “,” “\t”.
<lifehistory>	
<population>	
<speciesColumn>	
<dateFormat>	
<customTags>	This optional field allows for optional information to be stored in the database.

Lifehistory:

<name>	The name (including path from the configuration file) where the file can be found.
<alias>	An alias for the file.
<separator>	The character separating the columns in the datafile, e.g. “,” “\t”.

population:

<name>	The name (including path from the configuration file) where the file can be found.
<alias>	An alias for the file.
<separator>	The character separating the columns in the datafile, e.g. “,” “\t”.

LocationFile:

<name>	The name (including path from the configuration file) where the file can be found.
<alias>	An alias for the file.
<separator>	The character separating the columns in the datafile, e.g. “,” “\t”.
<locationIdColumn>	The column in the file containing the id of the location.
<eastingColumn>	The column in the file containing the easting coordinate. Coordinates aren’t strictly adhered to in Broadwick so a simple y-coordinate is sufficient.
<northingColumn >	The column in the file containing the northing coordinate. Coordinates aren’t strictly adhered to in Broadwick so a simple x-coordinate is sufficient.
<dateFormat>	
<customTags>	This optional field allows for optional information to be stored in the database.

TestsFile:

<name>	The name (including path from the configuration file) where the file can be found.
<alias>	An alias for the file.
<separator>	The character separating the columns in the datafile, e.g. “,” “\t”.
<idColumn>	One of these is required, specifying whether the test is performed on an individual, group (e.g. herd) or location (must match the id in the Location file).
<groupIdColumn>	
<locationIdColumn>	
<testDateColumn>	
<postiveResultColumn>	
<negativeResultColumn>	
<dateFormat>	
<customTags>	This optional field allows for optional information to be stored in the database.

A simplified configuration file is generated in the skeleton project. It contains configuration items for logging to console and to file for different logging levels (info, warning, error, debug, trace) and we can specify the pattern to apply to the log message.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<project>
  <logs>
    <console>
      <level>info</level>
      <pattern>[%thread] %-5level %msg %n</pattern>
    </console>
    <file>
      <name>broadwick.stochasticsir.log</name>
      <level>info</level>
      <pattern>[%thread] %-5level %msg %n</pattern>
      <overwrite>true</overwrite>
    </file>
  </logs>

  <models>
    <model id="Broadwick Project">
      <classname>broadwick.stochasticsir.StochasticSIR</classname>
    </model>
  </models>
</project>
```

Common logging patterns are:

%C	Outputs the fully-qualified class name of the caller issuing the logging request.
%M { %method }	Outputs the method name where the logging request was issued.
%L { %line }	Outputs the line number from where the logging request was issued.
%F { %file }	Outputs the file name of the Java source file where the logging request was issued. This is not very fast and should be avoided.
%d	Used to output the date of the logging event e.g. %d{HH:mm:ss,SSS}
%m (%msg)	Outputs the application-supplied message associated with the logging event.
%t (%thread)	Outputs the name of the thread that generated the logging event.
%n	Outputs the platform dependent line separator character or characters
%r	Outputs the number of milliseconds elapsed since the start of the application until the creation of the logging event.
%p { %level }	Outputs the level of the logging event.

More details on logging patterns can be found at <http://logback.qos.ch/manual/layouts.html>.

The model section requires a <classname> giving the fully qualified class name and optional <priors> and <parameter> sections.

Extending the Model

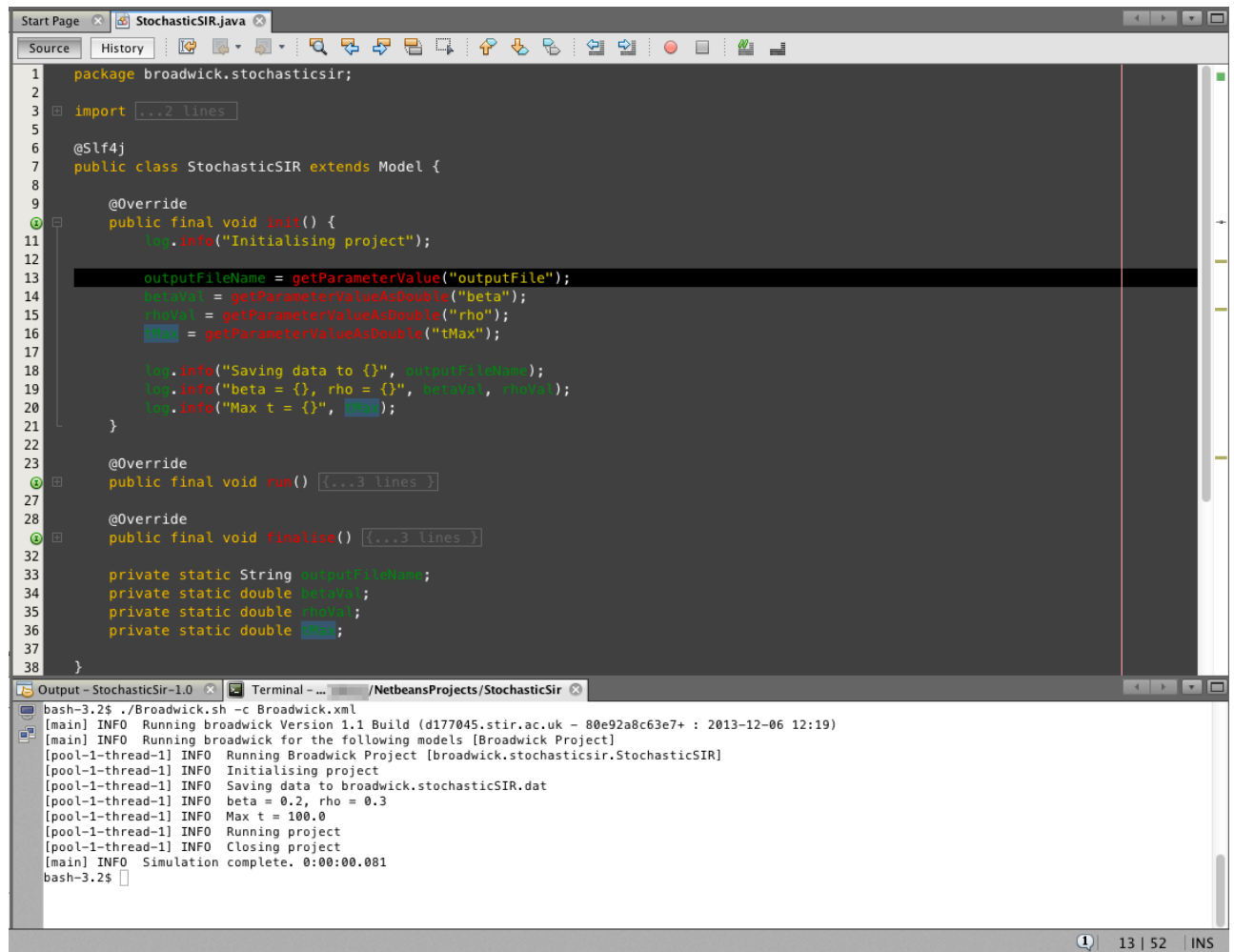
Our stochastic SIR model that we have created is a valid Broadwick model but does not perform any useful calculations. We will add some parameters to the configuration file and read (and log them) in the init() method.

Firstly, let us define beta and rho parameters for the susceptible->infectious rate and for the infectious->recovered rates respectively and parameters for the maximum time for which we will run the simulation and the name of a file in which we will save the time series data. To do this modify the configured model section by:

```
<model id="Broadwick Project">
  <classname>broadwick.stochasticsir.StochasticSIR</classname>

  <parameter id="beta" value="0.2" />
  <parameter id="rho" value="0.3" />
  <parameter id="tMax" value="100" />
  <parameter id="outputFile" value="broadwick.stochasticSIR.dat" />
</model>
```

Now edit the init() method of the StochasticSir class as shown below:



```
1 package broadwick.stochasticsir;
2
3 import ...2 lines
4
5
6 @Slf4j
7 public class StochasticSIR extends Model {
8
9     @Override
10     public final void init() {
11         log.info("Initialising project");
12
13         outputFileName = getParameterValue("outputFile");
14         beta = getParameterValueAsDouble("beta");
15         rho = getParameterValueAsDouble("rho");
16         tMax = getParameterValueAsDouble("tMax");
17
18         log.info("Saving data to {}", outputFileName);
19         log.info("beta = {}, rho = {}", beta, rho);
20         log.info("Max t = {}", tMax);
21     }
22
23     @Override
24     public final void run() {...3 lines }
25
26     @Override
27     public final void finalise() {...3 lines }
28
29     private static String outputFileName;
30     private static double beta;
31     private static double rho;
32     private static double tMax;
33 }
34
35
36
37
38
```

```
bash-3.2$ ./Broadwick.sh -c Broadwick.xml
[main] INFO Running broadwick Version 1.1 Build (d177045.stir.ac.uk - 80e92a8c63e7+ : 2013-12-06 12:19)
[main] INFO Running broadwick for the following models [Broadwick Project]
[pool-1-thread-1] INFO Running Broadwick Project [broadwick.stochasticsir.StochasticSIR]
[pool-1-thread-1] INFO Initialising project
[pool-1-thread-1] INFO Saving data to broadwick.stochasticSIR.dat
[pool-1-thread-1] INFO beta = 0.2, rho = 0.3
[pool-1-thread-1] INFO Max t = 100.0
[pool-1-thread-1] INFO Running project
[pool-1-thread-1] INFO Closing project
[main] INFO Simulation complete. 0:00:00.001
bash-3.2$
```

The 'Model' class contains `getParameterValue(String)`, `getParameterValueAsDouble(String)`, `getParameterValueAsInteger(String)`, `getParameterValueAsBoolean(String)` methods to extract parameters from the configuration file as strings (default), doubles, integers and booleans (if the parameter is written as "true" or "false").

Packages

Algorithms

Appendix

Maven as a Build Tool

There is no requirement to use maven as a build tool but as Broadwick and it's examples are built using maven this section will give a brief outline of how maven is used to create a simple model.

Maven uses an xml file to describe the classes to be built as well as the dependencies, dynamically downloading required libraries as needed. It uses the 'convention over configuration' paradigm imposing the directory structure given in the table below.

Directory	Purpose
Project home	Contains the pom and all the subdirectories.
src/main/java	Contains the java source code for the project.
src/main/resources	Contains the xsd file for configuring the project.
src/test/java	Contains any [JUnit or TestNG] test cases for the project.
src/test/resources	Contains resources necessary for testing.

Maven's equivalent to a makefile or Ant's build.xml is a 'project object model' which is stored in a pom.xml file. A good reference for the maven pom is <http://maven.apache.org/pom.html>.