

---

# **FastSTI Documentation**

***Release 0.2.2***

**Nathan Geffen and Stefan Scholz**

**Nov 21, 2019**



---

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Installation</b>	<b>3</b>
2.1	Linux etc. . . . .	3
2.2	Mac . . . . .	3
2.3	Windows . . . . .	4
2.4	On all machines . . . . .	4
<b>3</b>	<b>Getting started</b>	<b>7</b>
3.1	Simulations . . . . .	7
3.2	Example . . . . .	9
<b>4</b>	<b>Tutorial: Creating simulations</b>	<b>13</b>
4.1	Initial population . . . . .	13
4.2	Agent behaviour . . . . .	14
4.3	Reporting . . . . .	14
<b>5</b>	<b>Interesting example</b>	<b>17</b>
<b>6</b>	<b>Running FastSTI from Python</b>	<b>21</b>
<b>7</b>	<b>Input file formats</b>	<b>23</b>
7.1	Configuration . . . . .	23
7.2	Datasets . . . . .	24
7.3	Agents . . . . .	27
<b>8</b>	<b>Output file formats</b>	<b>29</b>
8.1	Results reports . . . . .	29
8.2	Agent output . . . . .	33
<b>9</b>	<b>Configuration file parameters</b>	<b>35</b>
9.1	after_events . . . . .	35
9.2	age_alpha . . . . .	35
9.3	age_beta . . . . .	36
9.4	age_input_time_step . . . . .	36
9.5	age_max . . . . .	36
9.6	age_min . . . . .	36
9.7	agents_input_file . . . . .	37

9.8	agents_output_file	37
9.9	before_events	37
9.10	birth_event_every_n	37
9.11	birth_rate	38
9.12	csv_delimiter	38
9.13	dataset_birth_infect	38
9.14	dataset_birth_resistant	39
9.15	dataset_birth_treated	39
9.16	dataset_coinfect	40
9.17	dataset_gen_infect	40
9.18	dataset_gen_mating	40
9.19	dataset_gen_resistant	41
9.20	dataset_gen_sex	41
9.21	dataset_gen_sex_preferred	41
9.22	dataset_gen_treated	42
9.23	dataset_infect	42
9.24	dataset_infect_stage	42
9.25	dataset_mortality	44
9.26	dataset_rel_period	44
9.27	dataset_single_period	44
9.28	during_events	45
9.29	report_frequency	45
9.30	initial_infect_stage	45
9.31	match_k	46
9.32	mutual_csv_partners	46
9.33	num_agents	46
9.34	num_simulations	47
9.35	partnerships_file	47
<b>10</b>	<b>Events</b>	<b>49</b>
10.1	_read_agents	50
10.2	_generate_agents	50
10.3	_age	51
10.4	_death	51
10.5	_initial_mating	52
10.6	_initial_rel	53
10.7	_mating_pool	53
10.8	_breakup	54
10.9	_shuffle_living	54
10.10	_shuffle_mating	54
10.11	_rkpm	55
10.12	_breakup_and_pair	56
10.13	_generate_and_pair	56
10.14	_infect	56
10.15	_stage	57
10.16	_coinfect	57
10.17	_birth	57
10.18	_report	58
10.19	_write_results_csv_header	58
10.20	_write_agents_csv	59
10.21	_write_agents_csv_header	59
10.22	_write_living_agents_csv	59
10.23	_write_dead_agents_csv	59
10.24	_write_partnerships_csv_header	60

<b>11 Extending FastSTI</b>	<b>61</b>
<b>12 Help improve FastSTI</b>	<b>63</b>
<b>13 Bugs and other issues</b>	<b>65</b>
<b>14 References</b>	<b>67</b>
<b>15 License</b>	<b>69</b>
<b>16 Credits</b>	<b>71</b>



# CHAPTER 1

---

## Introduction

---

FastSTI is a framework for implementing agent-based models of sexually transmitted infection epidemics.

It is designed to handle up to tens of millions of agents on a mid-range laptop, or to run many smaller simulations consisting of thousands of agents quickly in parallel. On a high performance computer it can run dozens of large simulations or hundreds of small ones in parallel.

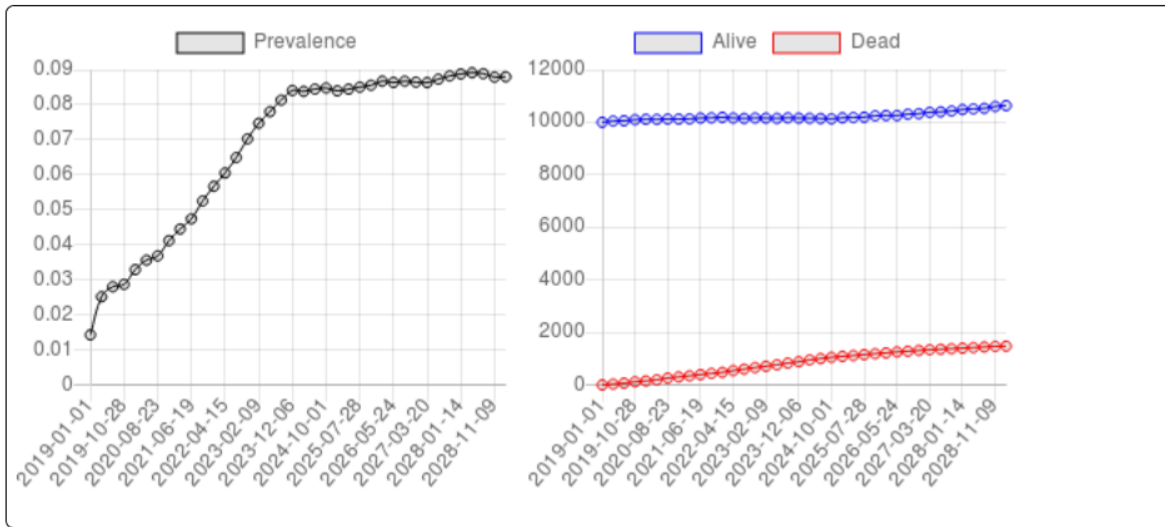
The output of the simulations is written to CSV files, which can be further processed in scripting languages like Python or R.

You can do sophisticated simulations without making any modifications to the code. But if you do wish to extend the framework, it is written in C. The framework has been designed with extensibility in mind.

FastSTI is developed and tested on GNU/Linux using the gcc compiler (and often the Clang compiler too). It uses two portable 3rd-party libraries: the GNU Scientific Library and GLib. The code adheres to the C11 standard and should compile on any standards-compliant modern C compiler.

FastSTI is free software (GPL licensed). If you are a C programmer, you can clone the [source code on Github](#) and modify it.

There's a web interface to a toy version of FastSTI on [Simhub](#) that you can play around with to get a feel for how it works. Here's a screenshot from Simhub:



**Warning:** FastSTI is in active development. It is usable (we think) but this documentation is changing almost daily. At this point no backwards compatibility is being implemented, and no warnings of incompatible changes are being documented. FastSTI is not yet at an alpha release stage. When it is more stable, we will remove this warning.

# CHAPTER 2

---

## Installation

---

FastSTI is easiest to run in a Posix environment (GNU Linux, FreeBSD, Unix etc), because we develop on Linux. Nevertheless, it isn't difficult to get it running on a Mac or Windows machine.

### 2.1 Linux etc.

1. FastSTI requires a standard C development environment, the [GNU Scientific Library](#) and [Gnome's Glib utility library](#).

On Ubuntu, Debian, Mint etc. this should do the trick:

```
sudo apt install build-essential libgsl-dev libglib2.0-dev meson
```

2. Download FastSTI from <https://www.simhub.online/media/dist/faststi/faststi-0.2.2.tar.gz> and decompress it.  
E.g.

```
wget https://www.simhub.online/media/dist/faststi/faststi-0.2.2.tar.gz; tar xzf faststi-0.2.2.tar.gz
```

This should create a directory called faststi-0.2.2 which contains the FastSTI files.

Now go to step 3.

### 2.2 Mac

1. Download and decompress this file: <https://www.simhub.online/media/dist/faststi/latest.tar.gz>

2. Using [Homebrew](#) install the dependencies:

```
brew install gcc pkg-config glib gsl meson
```

This should create a directory called faststi-0.2.2 which contains the FastSTI files.

Now go to step 3.

## 2.3 Windows

Under Windows you either need to install the Cygwin or MinGW environment, or if you are using Windows 10, you can install the Windows Subsystem for Linux. But to date we have only tested installation with Cygwin.

### 2.3.1 Cygwin

1. Use the Cygwin setup tool to install these packages:

- gcc-core
- make
- meson
- libgsl-dev
- libglib2.0-devel
- wget

2. After installation open the Cygwin terminal and run these steps:

```
wget https://www.simhub.online/media/dist/faststi/latest.tar.gz
tar xzvf latest.tar.gz
```

This should create a directory called faststi-0.2.2 which contains the FastSTI files.

Now go to step 3.

## 2.4 On all machines

3. Change into the directory that you've decompressed the distribution into: faststi-0.2.2

4. Now use the meson build system:

```
meson --buildtype release release
cd release
ninja
```

5. Then depending on your system either one of these should install the files:

```
sudo ninja install
```

Or:

```
ninja install
```

6. Now test that it works:

```
FSTI_DATA=../data faststi -t
```

This will run the test suite. If everything is properly installed then no failures should be reported.

---

**Note:** You may be wondering why we prefix the executable with *FSTI\_DATA=../data*. The *FSTI\_DATA* environment variable simply tells FastSTI where the data files are stored. As you work on FastSTI you may use a different directory for your data files. It's simply that the default data files are in the *data* directory.

---

To uninstall FastSTI simply go into the release directory and run:

```
sudo ninja uninstall
```



FastSTI takes as input a configuration file which tells it how many simulations to run, and how to run each simulation. It also takes data as input in files that we call datasets. And optionally, it takes a file of agents, though the agents can be generated by FastSTI itself, via instructions in the configuration file.

The outputs are a report of simulation results, and the agents.

### 3.1 Simulations

A simulation continuously iterates over sets of agents, executing events on the agents on each iteration (which we call a time step). The structure of a FastSTI simulation is:

```
Execute events before simulation runs
for each time-step
  for each event E
    for each agent A
      if E should be applied to A
        apply E to A
Execute events after simulation runs
```

The number of agents and the specific events to execute are specified in a configuration file. FastSTI's configuration file uses the *.ini* format, which are the standard simple configuration format used on MS Windows and the GTK framework popular on Linux systems.

You can configure the number of agents, the events and the order of events that execute upon them, the size of the time step (default 1 day), the number of time steps (default 10 years) and much else (see [Configuration file parameters](#)).

FastSTI has a number of useful built-in events useful for modelling STI epidemics (see [Events](#)). These include agent ageing, death, matching agents in sexual relationships, infection with the STI, disease advance, co-infection, and breakups.

There are also useful supporting events that read in agent files or generate the agents, write the agents to a CSV file, and write basic statistics to a CSV file.

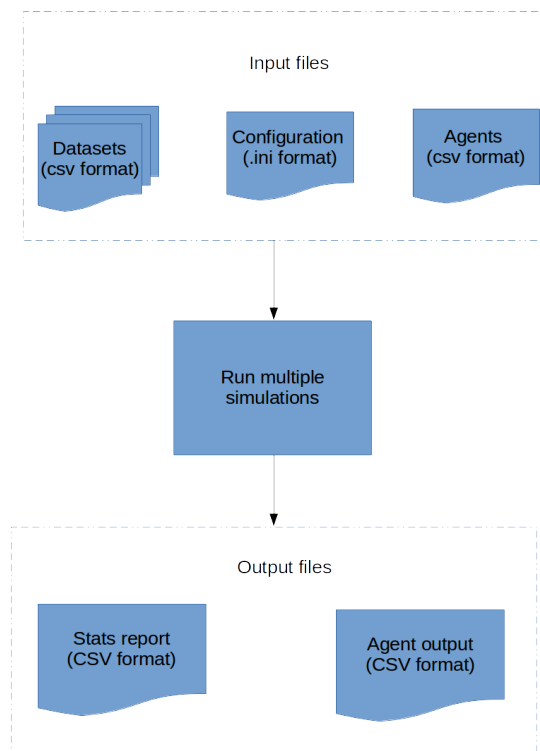


Fig. 1: FastSTI inputs include a configuration file, datasets and agent files. The outputs are an agent file and simulation results.

If you need more events, the framework has been designed with extensibility in mind. You can define new events in C, identify them to FastSTI, quickly recompile the code and use them.

## 3.2 Example

Let's start off with the simplest simulation. Change into the simulations/examples directory. Take a look at `egl.ini`.

```
# Faststi "Hello world" equivalent simulation
[Simulation_0]
after_events=_report
```

The first line is a comment.

The second line is the name of the simulation group: *First simulation*. A simulation group can have one or more simulations. This particular group has only one simulation.

The third line is one of the dozens of parameters used to configure simulations. The *after\_events* parameter tells FastSTI what events to execute when the simulation is finished. *\_report* is a built-in event that prints out basic information about the state of a simulation. All built-in events are prefixed with an underscore, to differentiate them from ones you might code yourself.

To run the simulation:

```
FSTI_DATA=../../data faststi -f egl.ini
```

The output may look something like this:

```
First simulation;0;0;2028-01-01;MIN_AGE_ALIVE;nan
First simulation;0;0;2028-01-01;MAX_AGE_ALIVE;nan
First simulation;0;0;2028-01-01;MEAN_AGE_ALIVE;nan
First simulation;0;0;2028-01-01;MEDIAN_AGE_ALIVE;nan
First simulation;0;0;2028-01-01;INFECT_RATE_ALIVE;-nan
First simulation;0;0;2028-01-01;POP_ALIVE;0
First simulation;0;0;2028-01-01;NUM_PARTNERS;0
First simulation;0;0;2028-01-01;MIN_AGE_DEAD;nan
First simulation;0;0;2028-01-01;MAX_AGE_DEAD;nan
First simulation;0;0;2028-01-01;MEAN_AGE_DEAD;nan
First simulation;0;0;2028-01-01;INFECT_RATE_DEAD;-nan
First simulation;0;0;2028-01-01;POP_DEAD;0
First simulation;0;0;2028-01-01;INITIAL_INFECTIIONS;0
First simulation;0;0;2028-01-01;SIMULATION_INFECTIIONS;0
First simulation;0;0;2028-01-01;INITIAL_MATCHES;0
First simulation;0;0;2028-01-01;SIMULATION_MATCHES;0
First simulation;0;0;2028-01-01;BREAKUPS;0
First simulation;0;0;2028-01-01;TIME_TAKEN;0
```

Note that it's in csv format, so you easily import it into Python or R and process it. You can also redirect the output to a file instead of standard output with the *results\_file* parameter.

The fields of the csv file are: the name of the simulation, the number of the current simulation, the number of the simulation within the current simulation group, the date within the simulation for which the output applies, a description field, and the value of the description field. E.g. the last two columns of the last line are `TIME_TAKEN` and `0`. This tells you that it took zero seconds for the simulation to run. Likewise the `POP_ALIVE` and `POP_DEAD` entries tell us that the population alive and dead in this simulation on 1 January 2028 is 0.

Prefixing every run with setting the `FSTI_DATA` directory is tedious. Let's fix that. In a POSIX environment (Linux, FreeBSD etc) that's easy. Change into the `faststi-0.2.2` directory. Now set and export the `FSTI_DATA` environment

variable like this:

```
export FSTI_DATA=`pwd`/data
```

Test that it's working:

```
faststi -t
```

If no errors are reported, then it's working. But at present you'll have to execute the export command every time you open a terminal. To make it permanently part of your environment, place it in your .profile or .bashrc scripts, depending on the flavour of POSIX you're using.

### 3.2.1 A full simulation

The output of the eg1.ini simulation is rather uninteresting. To get more interesting output we need a more interesting simulation. Take a look at eg2.ini.

```
1  # First interesting simulation
2
3  [Full simulation]
4
5  num_simulations=4
6  num_agents=10000
7  time_step=1 DAY
8  simulation_period=10 YEARS
9
10 before_events=_write_agents_csv_header;_generate_and_pair;_report;_write_agents_csv
11 during_events=_age;_breakup_and_pair;_infect;_stage;_birth;_death
12 after_events=_write_agents_csv;_report
13
14 dataset_gen_sex=dataset_gen_sex.csv
15 dataset_gen_sex_preferred=dataset_gen_sex_preferred.csv
16 dataset_gen_infect=dataset_gen_infect.csv
17 dataset_gen_treated=dataset_gen_treated.csv
18 dataset_gen_resistant=dataset_gen_resistant.csv
19 dataset_gen_mating=dataset_gen_mating.csv
20
21 dataset_birth_infect=dataset_gen_infect.csv
22 dataset_birth_treated=dataset_birth_treated.csv
23 dataset_birth_resistant=dataset_birth_resistant.csv
24
25 dataset_rel_period=dataset_rel.csv
26 dataset_single_period=dataset_single.csv
27 dataset_infect=dataset_infect.csv
28 dataset_infect_stage=dataset_infect_stage.csv
29 dataset_mortality=dataset_mortality_simple.csv
30
31
32 agents_output_file=agents_out.csv
33 results_file=results.csv
34
35 threads=1
36
37 [Change time period]
38
39 threads=0 # As many threads as there are cores will execute
40 simulation_period=5 YEARS
```

Assuming you are in the simulations/examples directory you can run this simulation as follows:

```
faststi -f eg2.ini
```

This will take about 10 seconds to run, depending on your PC.

This is what the configuration does:

- Line 1 is a comment. Comments start with a #.
- Line 3 specifies the name of the first group of simulations: *Full simulation*
- Line 5 specifies the number of simulations to run in this group. Note that parameter names, like *num\_simulations*, are case-sensitive.
- Line 6 specifies the number of agents in the simulation.
- Line 7 specifies the time period represented by each simulation iteration, 1 day in this case. The format for this parameter and others like it that specify a time period is a positive integer followed by either MINUTE, HOUR, DAY, WEEK, MONTH or YEAR. You can also use the plural of any of these time periods or any combination of lower and upper case (i.e. minute, minutes, hour, hours, day, days, week, weeks, month, months or year, years). You can also put a hyphen instead of a space between the integer and the time period. If you leave out the time period, it's assumed to be minutes. In FastSTI, the predefined time period have the following number of minutes:
  - hour: 60
  - day: 1,440
  - week: 10,080
  - month: 43,830
  - year: 525,949
- Line 8 specifies the simulation period: 10 years in this case. The number of time steps (or iterations) of the simulation is calculated by dividing the simulation\_period by the time\_step. In this simulation there are therefore 3,652 iterations:  $10 \times 525949 / 1440$ .
- Line 10 lists the events that are to be run before each simulation starts.
  - The `_write_agents_csv_header` event simply writes the first line of the csv file to which agents will be written.
  - The `_generate_and_pair` event generates agents (equal to the *num\_agents* parameter) and pairs a subset of them in sexual relationships.
  - The `_report` event prints out some simple statistics about the agent population.
  - The `_write_agents_csv` event prints out the agents in csv format.
- Line 11 lists the events that are run on every iteration of the simulation.
  - The `_age` event increases the age of each agent by the *time\_step* of the simulation (one day in this example).
  - The `_breakup_and_pair` event pairs a subset of agents into sexual partnerships and breaks up another subset of agents who are already in sexual partnerships.
  - The `_infect` event infects a subset of agents in sexual partnerships with other infected agents with the pathogen being studied, e.g. HIV.
  - The `_stage` event manages the infection progression of infected agents. For example, agents with HIV will first be in an acute sero-conversion phase, than a chronic infection stage, and then one or more stages that are analogous to progression to AIDS.
  - The `_birth` and `_death` events create new agents and kill agents respectively.

Many of these events depend on pre-specified parameters to calculate probabilities of the events occurring and, perhaps, other outcomes. These data are described in dataset files which are described in the [Datasets](#) section.

- Lines 14 to 29 list the names of the datasets associated with the various events. For example the *dataset\_mortality* parameter tells the simulation the probability of an agent with a given set of characteristics dying.
- Line 32 tells the simulation to write agents out to a csv file called *agents\_out.csv*.
- Line 33 tells the simulation to write simple population level statistical information produced by the *\_report* event to a file called *results.csv*.
- Line 35 tells the simulation to run as a single thread. In other words each of the four simulations (specified on line 5) will run consecutively.
- Line 37 specifies a new simulation group called *Change time period*.
- Line 39 tells FastSTI to run the simulations in parallel, using up to as many threads as there are CPU cores in the machine. On a quad-core CPU, all four simulations could run at the same time.
- Line 40 specifies a different simulation period (5 years) to the *Full simulation* set of simulations. All other parameters set in the *Full simulation* set carry through to this simulation set.

There are more configuration examples in the simulation directory.

---

## Tutorial: Creating simulations

---

When you create an agent-based model, there are generally three things to consider: creating an initial population, agent behaviour during the simulation and reporting.

### 4.1 Initial population

FastSTI allows you to generate the initial population or to read agents in from a file.

The *before\_events* parameter tells FastSTI how to initialize the agents.

#### 4.1.1 Generating the population

The easiest way to generate a population is to use the *\_generate\_and\_pair* event. E.g. in the simulation configuration file there would be a line like this:

```
before_events = _generate_and_pair; <other events to run before the simulation>
```

The *\_generate\_and\_pair* event is a composite event which calls five other events: *\_generate\_agents*, *\_initial\_mating*, *\_shuffle\_mating*, *\_rkpm* and *\_initial\_rel* events.

Together these events create the initial agents in the simulation, and place some of them into relationships. Click on each of these events to see how they work and how you can specify the number of agents, the ages of the agents, their sexes, sexual preferences and the distribution of the relationships in the set of initial agents.

#### 4.1.2 Reading an agent file

Reading in an agent file perhaps gives greater flexibility than the built-in agent generating functionality of FastSTI.

The file would typically be created in a scripting language like R or Python or even be hand-crafted.

The *\_read\_agents* event is used to read in the agents. In the simulation configuration file there would be a line like this:

```
before_events = _read_agents; <other events to run before the simulation>
```

Read the [section on agent input formats](#) to find out how to set up the agent input CSV file.

## 4.2 Agent behaviour

In FastSTI agent behaviour takes place in events. There are predefined events that provide for agents to *age*, *form sexual relationships with other agents and break-up with them*, *become infected*, *start treatment* <stage\_ref>, *change their infection status* <stage\_ref> (e.g. become sicker or become better), be *born*, and *die*.

Most of these events are usually executed using the [during\\_events](#) parameter. Although it's conceivable you may also need to execute agent behaviour before and after a simulation using the [before\\_events](#) and [after\\_events](#) parameters respectively.

If these events don't meet your needs the FastSTI framework allows you to *code your own events* in C.

## 4.3 Reporting

Usually, you need statistics about the simulation before, during and after the simulation has run. FastSTI has a reporting facility to do write out demographic and statistical information at any stage of a simulation.

Reports are written to a single output file in CSV format. If you're running multiple simulations, the report lines from each simulation will be interleaved with each other throughout the file. But each report line has enough information for you to know which simulation produced it.

You'll usually need to use a spreadsheet program or a scripting language like R or Python to process the output CSV file. See the scripts directory for examples of report processing code that you make using FastSTI much easier for you.

Reporting is just another event in FastSTI. There are a few provided: [\\_report](#) - which outputs statistics, [\\_write\\_agents\\_csv](#) - which outputs every agent in the simulation, [\\_write\\_living\\_agents\\_csv](#) - which outputs every living agent in the simulation, and [\\_write\\_dead\\_agents\\_csv](#) - which outputs every dead agent in the simulation. There are also events to write the headers for the output CSV files.

The [report\\_frequency](#) parameter specifies how frequently to run these events during a simulation.

Use the events that output agents carefully. If you're executing [\\_write\\_agents\\_csv](#) during the simulation, having a report\_frequency of 1 will slow your simulation down and use huge amounts of hard drive space.

A typical report setup for a simulation looks something like this:

```
results_file = results.csv
agents_output_file = agents_out.csv
report_frequency = 365
before_events = _write_agents_csv_header; _write_results_csv_header; <other
events>; _report
during_events = <other events>; _report; _write_living_agents_csv
after_events = <other events>; _report; write_agents_csv
```

If you don't need to know details about individual agents, then it's best not to execute the events that output agents. Your simulation will be faster and use much less disk space. Then you can also have much more frequent report writing. E.g.

```
results_file = results.csv
report_frequency = 10
```

(continues on next page)

(continued from previous page)

```
before_events = _write_results_csv_header; <other
events>; _report
during_events = <other events>; _report
after_events = <other events>; _report
```



## CHAPTER 5

---

### Interesting example

---

Here is an experiment that we hope demonstrates the versatility of FastSTI.

It shows some interesting and counter-intuitive results, as well as how to use FastSTI in conjunction with a scripting language like R.

Here is the input configuration, which you can also find in `simulations/examples/experiment.ini`:

```
1  [Simulation Group 0]
```

```
2
3  threads=0
4  match_k = 1
5  num_simulations=30
6  num_agents=10000
7  time_step=1 DAY
8  simulation_period=10 YEARS
9  report_frequency=100
10
11 before_events=_write_results_csv_header;_generate_and_pair;_report
12 during_events=_age;_breakup_and_pair;_infect;_stage;_birth;_death;_report
13 after_events=_report
14
15 dataset_gen_sex=dataset_gen_sex.csv
16 dataset_gen_sex_preferred=dataset_gen_sex_preferred.csv
17 dataset_gen_infect=dataset_gen_infect.csv
18 dataset_gen_treated=dataset_gen_treated.csv
19 dataset_gen_resistant=dataset_gen_resistant.csv
20 dataset_gen_mating=dataset_gen_mating.csv
21
22 dataset_birth_infect=dataset_gen_infect.csv
23 dataset_birth_treated=dataset_birth_treated.csv
24 dataset_birth_resistant=dataset_birth_resistant.csv
25
26 dataset_rel_period=dataset_rel.csv
27 dataset_single_period=dataset_single.csv
28 dataset_infect=dataset_infect.csv
```

(continues on next page)

(continued from previous page)

```

29 dataset_infect_stage=dataset_infect_stage.csv
30 dataset_mortality=dataset_mortality_simple.csv
31
32 results_file=results.csv
33
34 [Simulation Group 1]
35 match_k = 100
36
37 [Simulation Group 2]
38 match_k = 1
39 num_agents=100000
40
41 [Simulation Group 3]
42 match_k = 100

```

It tells FastSTI to run four groups of simulations. Each group runs 30 simulations.

All groups run daily for ten years starting on 1 January 2018. For each simulation statistics are printed every 100 days, and then at the end, 1 January 2028. (The dates don't matter, by the way.)

- The first group (group 0) is 10,000 agents with random matching.
- The second group is 10,000 agents with k-nearest neighbour matching (using FastSTI's `_rkpm` event) x with `k=100`.
- The third group is 100,000 agents with random matching.
- The fourth group is 100,000 agents with k-nearest neighbour matching with `k=100`.

The results are written to a file called `results.csv`. The file is large so we've [compressed](#) it. On a mid-range laptop with an i5 with four cores and 12GB of RAM it took just over 16 minutes to run the 120 simulations.

We then ran `results.csv` through this R script, which you can find in `scripts/processResultsFile.R`. (The equivalent Python script is in `scripts/processResultsFile.py`.)

```

1  # Takes a results CSV file produced by FastSTI and runs a function
2  # (default mean) over a statistic (description) grouping the result
3  # either by the maximum date or all the dates.
4
5  processFastSTIResults <- function(filename="results.csv",
6                                   sep=";",
7                                   description="INFECT_RATE_ALIVE",
8                                   fun=mean,
9                                   header=TRUE,
10                                  filter_max_date=TRUE) {
11    inp = read.csv(filename, sep=sep, header=header)
12    values = inp[inp$description==description,]
13    if (filter_max_date) {
14      analysisDate = max(as.Date(values$date))
15      values = values[as.Date(values$date)==analysisDate,]
16    }
17    mean_values = aggregate(values$value,
18                           by=list(values$name,
19                                   values$date),
20                           FUN=fun)
21  }
22
23  print(processFastSTIResults())

```

This is the output:

1	Simulation Group 0	2028-01-01	0.4280027667
2	Simulation Group 1	2028-01-01	0.3647885667
3	Simulation Group 2	2028-01-01	0.4283530667
4	Simulation Group 3	2028-01-01	0.2342883667

What do the results tell us?

1. Random matching results in much higher infection rates than “intelligent” matching algorithms that try to pair agents using realistic criteria. This is expected. With random matching, one would expect an infection to spread quite rapidly through a population. With more realistic matching, we expect more sexual networks to form and the infection only occasionally moves from one network to another.
2. Random matching gives the same results irrespective of the number of agents (42.8%). This too is expected.
3. With “intelligent” matching prevalence decreases as the population increases (a mean of 36.5% for the 10,000 agent simulations after ten years vs 23.4% for the 100,000 agent simulations). This is not intuitive, at least not to us, though we have hypotheses for why it happens. We leave it to you to speculate further.



---

## Running FastSTI from Python

---

**Warning:** The Python interface to FastSTI is highly experimental. We hope to make Python the main way to use FastSTI because Python is so easy to use and popular among science researchers. But at this point we've only got the bare basics working. The entire Python interface to FastSTI is likely to change. Also currently if FastSTI exits because of an error, the Python interpreter also exits instead of throwing an exception, which is annoying. We will fix this in due course.

A basic Python 3 interface is supported (we don't support Python 2 and don't intend to). You'll need to install [pandas](#).

In the scripts directory of the FastSTI folder is `faststi.py`.

To see the command line options it supports run (for example):

```
python faststi -h
```

To execute a set of simulations:

```
python faststi -c my_simulations.ini
```

Example of how to use `faststi` in a Python environment:

```
import faststi
import pandas

simset1 = faststi.SimulationSet(config_filename="my_simulations.ini")

# Runs all the simulations and save the results into a dataframe
dataframe = simset1.run()

# Slightly more sophisticated example
simset2 = faststi.SimulationSet(config_filename="my_simulations.ini")
print("There are", len(simset2.simulations), "simulations.")
# Change the match_k parameter in the third simulation (0-based indexing)
simset2.simulations[2].set_parm("match_k", 300)
```

(continues on next page)

(continued from previous page)

```
# Run the first four simulations: 0,1,2,3
dataframe2 = simset2.run(0, 4)
# Run the remaining simulations
dataframe3 = simset2.run(4)
```

In time, we hope to make this much more comprehensive, robust and useful.

## 7.1 Configuration

The configuration file, which specifies the simulations, is in Windows ini format. Here is an example:

```

1  # Faststi test configuration file
2
3  [Simulation 0]
4
5  num_simulations=1
6  num_agents=10000
7  simulation_period=10 YEARS
8
9  agents_output_file=agents_out.csv
10 results_file=results.csv
11
12 dataset_gen_sex=dataset_gen_sex.csv
13 dataset_gen_sex_preferred=dataset_gen_sex_preferred.csv
14 dataset_gen_infect=dataset_gen_infect.csv
15 dataset_gen_treated=dataset_gen_treated.csv
16 dataset_gen_resistant=dataset_gen_resistant.csv
17 dataset_gen_mating=dataset_gen_mating.csv
18
19 dataset_birth_infect=dataset_gen_infect.csv
20 dataset_birth_treated=dataset_birth_treated.csv
21 dataset_birth_resistant=dataset_birth_resistant.csv
22
23 dataset_rel_period=dataset_rel.csv
24 dataset_single_period=dataset_single.csv
25 dataset_infect=dataset_infect.csv
26 dataset_infect_stage=dataset_infect_stage.csv
27 dataset_mortality=dataset_mortality_simple.csv
28
29 before_events=_write_agents_csv_header;_generate_and_pair;_report;_write_agents_csv

```

(continues on next page)

(continued from previous page)

```

30 during_events=_age;_breakup_and_pair;_infect;_stage;_birth;_death
31 after_events=_write_agents_csv;_report
32
33 match_k=100
34 threads=1
35
36 [Simulation 1]
37
38 dataset_mortality=dataset_mortality_complicated.csv
39 match_k=300

```

- Comment lines begin with a #.
- A # in the middle of a line also denotes a comment until the end of the line.
- A simulation group name is enclosed in square brackets `[]`.
- Within a simulation group, parameters are specified as key-value pairs. A key is separated from its value by an equals sign. White space before and after the = sign is ignored.
- Some parameters (e.g. `during_events`) take multiple values. Each value must be separated by a semi-colon `;`. White space before or after the semi-colon is ignored.
- Blank lines are ignored.
- Each key must be a predefined parameter. If you use a key that isn't a predefined parameter, the FastSTI will give an error message and terminate. To see all the parameters, run:

```
faststi -p
```

- Key-values set in a simulation group are carried through to subsequent groups, unless they are specified again in the new group. E.g, in the example above, `dataset_mortality` and `match_k` are specified again for the simulation group called *Simulation 1*. All the other parameters are identical to *Simulation 0*.

Besides setting parameters in the input configuration file, you can modify them on the command line using this format:

```
faststi -c=<value>[;<value>]* -f <filename>
```

## 7.2 Datasets

Many events depend on datasets. The format of the dataset file is a little cumbersome but it is designed for a combination of speed and safety.

Datasets can either be placed in the `data` directory, or in the directory in which the simulation is being run. Alternately, the `FASTI_DATA` environment variable can be set to the location where the datasets are located.

Datasets are csv files. The default delimiter is a semi-colon, not a comma. You can change this by setting the `csv_delimiter`

A standard dataset consists of 0 or more columns representing the values of agent properties, followed by 1 or more columns representing probabilities or other derived values. Events typically match the properties of the current agent being operated upon to the corresponding row in the dataset in order to obtain the appropriate probability of the event occurring. Sometimes there

Let's start with the very simplest of the supplied datasets. It is `dataset_gen_sex.csv` and it is located in the `data` directory. It is used by the `_generate_agents` event to initialize the sex of agents at the beginning of simulations. Here it is:

```

1 Probability
2 0.5

```

Line 1 is simply the CSV header. It is called *Probability* here but we could have named it anything. Line 2 is 0.5, the odds of being male. The event uses this to set approximately half the agents to male and half to female when it generates agents.

Here's a more typical dataset, *dataset\_gen\_infect.csv*, also used by the *\_generate\_agents* event to determine the infection stage, if any, of agents when they are initialized.

```

1 sex;sex_preferred;age|10-YEAR;1;2;3;4|4
2 0;0;0;0;0;0;0
3 0;0;1;0;0;0;0
4 0;0;2;0.1;0.2;0.3;0.4
5 0;0;3;0.1;0.2;0.3;0.4
6 0;0;4;0.05;0.1;0.15;0.2
7 0;0;5;0.025;0.05;0.075;0.1
8 0;1;0;0;0;0;0
9 0;1;1;0;0;0;0
10 0;1;2;0.05;0.1;0.15;0.2
11 0;1;3;0.05;0.1;0.15;0.2
12 0;1;4;0.025;0.05;0.075;0.1
13 0;1;5;0.0125;0.025;0.0375;0.05
14 1;0;0;0;0;0;0
15 1;0;1;0;0;0;0
16 1;0;2;0.05;0.1;0.15;0.2
17 1;0;3;0.05;0.1;0.15;0.2
18 1;0;4;0.025;0.05;0.075;0.1
19 1;0;5;0.0125;0.025;0.0375;0.05
20 1;1;0;0;0;0;0
21 1;1;1;0;0;0;0
22 1;1;2;0.05;0.1;0.15;0.2
23 1;1;3;0.05;0.1;0.15;0.2
24 1;1;4;0.025;0.05;0.075;0.1
25 1;1;5;0.0125;0.025;0.0375;0.05

```

In the example HIV model provided, there are five possible values for *infection*:

- 0 = uninfected
- 1 = virally suppressed (usually on treatment)
- 2 = primary infection (highly infectious)
- 3 = chronic infection (usually lasts several years)
- 4 = Final stage (AIDS)

Let's start with the header (line 1). The last column is *4|4*. The first "4" is simply the name of the column (representing stage 4 infection), and could have been called anything. But the "4" after the pipe (|) tells FastSTI that the last four columns all represent probabilities. If a dataset contains more than one probability column then this must be specified. FastSTI then knows that the first three fields, *sex*, *sex\_preferred* and *age* are not probability columns, and correspond precisely to the names of fields in the *fsti\_agent* data structure. If they didn't, FastSTI would terminate with an error.

The dataset needs an entry (or row) for each combination of *sex*, *sex\_preferred* and *age*. Also the first row of every dataset after the header must start with every property set to 0, and then cycle incrementally through all combinations of possible values for the properties. This may sound tiresome, but it ensures that probabilities can be looked up using a random access search, rather than having to sequentially search the table.

There is one important short-cut. Notice the column headed "age|10-YEAR". The pipe followed by either an integer

or a time period, tells FastSTI to divide the agent's age by this number, in this case 10 years, in order to get the value to search for in the dataset. So an agent with age 45 will have its age divided by 10 which gives it a lookup value for its age of 4 (the .5 is dropped - this is integer division).

What about an agent whose age is 60 or more (because the ages run from 0 to 5)? The dataset lookup algorithm assumes any agent property greater than the largest value is equal to the largest value.

Consider an agent who is male, prefers to have sex with females and is 31 years old. What is the probability they are HIV-positive (in this dataset)? And if HIV-positive, what infection stage is he likely to be in?

The agent matches line 11, which corresponds to male agents (first column with a value of 0) whose preferred sexual partner is female (second column with a value of 1) and agents aged 30 to 39 (third column with a value of 3, i.e. 31 / 3).

To determine if the agent is infected with HIV, the `_generate_agents` event samples a uniform random number,  $r$ .

- If  $r$  is less than 0.05 (the value in column 4 of line 11), the agent is in stage 1.
- If  $r$  is less than 0.1 (the value in column 5 of line 11), the agent is in stage 2.
- If  $r$  is less than 0.15 (the value in column 6 of line 11) the agent is in stage 3.
- If  $r$  is less than 0.2 (the value in column 7 of line 11) the agent is in stage 4.
- Else if  $r$  is greater than or equal to 0.2, the agent is uninfected.

With most events, the agent characteristics you use are up to you. You could create a dataset for generating the initial infection status of agents that doesn't take into account `sex_preferred` or `age`. Alternately, you could add a `coinfection` column (because there is a field called `coinfection` in the FastSTI agent structure), and make the infection probabilities dependent on that.

There is somewhat less flexibility with the probability fields. These are event-specific. As it happens the code that sets the infection stage expects one or more user-defined stages, so you can specify fewer or more than the four stages in the above example.

### 7.2.1 Two-agent datasets

Some events need to make a decision based on two agents. In modelling sexually transmitted infections, the most obvious example is an event that determines if an agent becomes infected. FastSTI's supplied `_infect` event does just this. It iterates over all pairs of agents in sero-discordant sexual relationships, and determines whether the negative partners becomes infected.

Consider two agents,  $a$  and  $b$ . One,  $a$ , is uninfected, and the other  $b$  is infected. If we want the risk of infection to be determined by  $a$ 's sex and whether it is in a same-sex or opposite-sex relationship with  $b$  then we need some way of specifying this in a dataset. Also, we are interested in what infection stage  $b$  is in. If  $b$  is on treatment, for example, the risk of infecting  $a$  may be very low.

The `dataset_infect.csv` dataset shows how this is handled in FastSTI.

```
1 sex;sex|1|~;infected;probability
2 0;0;0;0
3 0;0;1;0
4 0;0;2;0.02
5 0;0;3;0.008
6 0;0;4;0.008
7 0;1;0;0
8 0;1;1;0
9 0;1;2;0.01
10 0;1;3;0.004
11 0;1;4;0.004
```

(continues on next page)

(continued from previous page)

```

12 1;0;0;0
13 1;0;1;0
14 1;0;2;0.012
15 1;0;3;0.005
16 1;0;4;0.005
17 1;1;0;0
18 1;1;1;0
19 1;1;2;0.0001
20 1;1;3;0.0001
21 1;1;4;0.0001

```

The header (line 1) contains two columns named *sex*. The first one corresponds to the uninfected agent, *a*. The second and third columns are the sex and infection stage of *b*. How does FastSTI know this? Look at the second column heading: *sex|1|~*. The first pipe (|) is used to separate sex from the amount the property must be divided by. Well, unlike age, we don't want the sex to be more granular, so we specify it as 1. The second pipe is followed by a tilde (~). The tilde in the column header tells FastSTI that this is a two-agent lookup table and the second agent's properties start in this column. So the second and third columns belong to agent *b*. The final column, with name *probability*, is simply the probability of becoming infected. (By default events are executed daily, so the probability must correspond to this time-step.)

So if agent *a* is a female, and agent *b* is male in stage 2 (primary infection), what is the risk of *a* becoming infected on this iteration of the *\_infect* event? The answer is given by line 14: 0.012.

## 7.3 Agents

Instead of generating agents, you can provide an agent file as input to the simulation. In fact, since the agent generation features of FastSTI are currently quite limited, you'll probably prefer to supply an agent file.

The agents must be specified in a CSV file. The column names in the header row must correspond to one or more field names in FastSTI's agent structure, which is declared as *struct fsti\_agent* in the source file *src/fsti-agent.h*. The fields are:

- *id*: unsigned 32 bit integer, unique for each agent (If you do not include this field, FastSTI automatically provides this value for each agent, starting from 0.)
- *sex*: unsigned 8 bit integer (0 is male, 1 is female. Higher values are user-defined.)
- Either *sex\_preferred* or *orientation*, an unsigned 8 bit integer (Do not use both fields. We recommend using *sex\_preferred* rather than *orientation*. For *sex\_preferred* 0 is male, 1 is female. For *orientation* either use 0 and 1 for heterosexual and homosexual respectively, or 0, 1, 2 and 3 for MSM, MSW, WSM and WSW respectively. Higher values are user-defined.)
- *age*: a positive year age of an agent between 0 and 120.
- *birthday*: a signed 32 bit integer (Unless you understand the internal workings of FastSTI very well, we recommend you rather use *age*)
- *infected*: unsigned 8 bit integer (0 is uninfected. 1 and up can correspond to stages of infection.)
- *treated*: unsigned 8 bit integer (0 is untreated. 1 and up can correspond to treatment regimens.)
- *resistant*: unsigned 8 bit integer (0 is no resistance. You can either use a simple approach to resistance, whereby 1 means resistant to treatment regimen 1, 2 to treatment regimen 2 etc, or you can use a more complex binary bitmask approach where 1 denotes resistance to regimen 1 only, 10, denotes resistance to regimen 2, 11 denotes resistance to regimen 1 and 2 etc.)

- `coinfected` (0 means not coinfecting. 1 and up denotes different types of coinfection as chosen by the user. Once again, as with the `resistant` field, either a simple or bitmask approach can be used.)
- `partners_0`, `partners_1`, and `partners_2`: unsigned 32 bit integers denoting the id of a sexual partner of this agent (-1 implies agent is single. The agents are typically numbered from 0. Note: None of the default FastSTI events currently caters for concurrency. Only use `partners_1` and `partners_2` if you are implementing events that rely on partner concurrency. If you need more partners, change the value of `FSTI_MAX_PARTNERS` in `fsti_userdefs.h`.)
- `relchange_0`, `relchange_1`, and `relchange_2`: unsigned 32 bit integers corresponding to the iteration (i.e. time step) in the simulation when the agent's relationship status for `partner_0`, `partner_1` and `partner_2` respectively should change, either to single for agents with partners or to be placed in the mating pool if the agent is single

Here is an example CSV file. The default delimiter is a semi-colon, not a comma. You can change this by setting the `csv_delimiter`.

```
1 id;age;infected;sex;sex_preferred;partners_0
2 0;45.21;0;1;0;-1
3 1;47.35;1;0;1;0
4 2;36.62;0;1;0;-1
5 3;35.40;0;1;0;-1
6 4;24.25;0;0;1;-1
7 5;24.12;0;0;1;4
8 6;23.26;0;0;1;-1
9 7;45.17;0;0;1;-1
10 8;34.81;0;0;1;-1
11 9;35.80;0;0;1;8
```

---

## Output file formats

---

### 8.1 Results reports

Whenever the *\_report* is executed, it writes output in CSV format either to the standard output device (default) or an output file as specified by the *results\_file* key in the configuration. This output is meant to be post-processed by you either in a spreadsheet, or using a scripting language like R or Python.

---

**Note:** The *\_report* event prints out quite basic information. If you feel comfortable writing C code, you can enhance it in the *fsti\_userdefs.h* file by defining the *FSTI\_HOOK\_REPORT* macro. See the *FSTI\_REPORT* macro in *fsti-defaults.h*.

---

Here is an example output from a set of simulations:

```

1  name;sim;num;date;description;value
2  Simulation_0;0;0;2018-01-01;MIN_AGE_ALIVE;15.000000
3  Simulation_0;0;0;2018-01-01;MAX_AGE_ALIVE;49.000000
4  Simulation_0;0;0;2018-01-01;MEAN_AGE_ALIVE;23.000000
5  Simulation_0;0;0;2018-01-01;MEDIAN_AGE_ALIVE;18.000000
6  Simulation_0;0;0;2018-01-01;INFECT_RATE_ALIVE;0.077900
7  Simulation_0;0;0;2018-01-01;POP_ALIVE;20000
8  Simulation_0;0;0;2018-01-01;NUM_PARTNERS;3573
9  Simulation_0;0;0;2018-01-01;MIN_AGE_DEAD;nan
10 Simulation_0;0;0;2018-01-01;MAX_AGE_DEAD;nan
11 Simulation_0;0;0;2018-01-01;MEAN_AGE_DEAD;nan
12 Simulation_0;0;0;2018-01-01;INFECT_RATE_DEAD;-nan
13 Simulation_0;0;0;2018-01-01;POP_DEAD;0
14 Simulation_0;0;0;2018-01-01;INITIAL_INFECTIONS;1558
15 Simulation_0;0;0;2018-01-01;SIMULATION_INFECTIONS;0
16 Simulation_0;0;0;2018-01-01;INITIAL_MATCHES;3573
17 Simulation_0;0;0;2018-01-01;SIMULATION_MATCHES;0
18 Simulation_0;0;0;2018-01-01;BREAKUPS;0
19 Simulation_0;0;0;2018-01-01;TIME_TAKEN;1

```

(continues on next page)

(continued from previous page)

```

20 Simulation_0;0;0;2028-01-01;MIN_AGE_ALIVE;15.000000
21 Simulation_0;0;0;2028-01-01;MAX_AGE_ALIVE;59.000000
22 Simulation_0;0;0;2028-01-01;MEAN_AGE_ALIVE;30.000000
23 Simulation_0;0;0;2028-01-01;MEDIAN_AGE_ALIVE;26.000000
24 Simulation_0;0;0;2028-01-01;INFECT_RATE_ALIVE;0.142421
25 Simulation_0;0;0;2028-01-01;POP_ALIVE;23592
26 Simulation_0;0;0;2028-01-01;NUM_PARTNERS;8442
27 Simulation_0;0;0;2028-01-01;MIN_AGE_DEAD;15.000000
28 Simulation_0;0;0;2028-01-01;MAX_AGE_DEAD;58.000000
29 Simulation_0;0;0;2028-01-01;MEAN_AGE_DEAD;31.000000
30 Simulation_0;0;0;2028-01-01;INFECT_RATE_DEAD;0.738192
31 Simulation_0;0;0;2028-01-01;POP_DEAD;741
32 Simulation_0;0;0;2028-01-01;INITIAL_INFECTIONS;1558
33 Simulation_0;0;0;2028-01-01;SIMULATION_INFECTIONS;2349
34 Simulation_0;0;0;2028-01-01;INITIAL_MATCHES;3573
35 Simulation_0;0;0;2028-01-01;SIMULATION_MATCHES;145574
36 Simulation_0;0;0;2028-01-01;BREAKUPS;140193
37 Simulation_0;0;0;2028-01-01;TIME_TAKEN;6
38 Simulation_0;1;1;2018-01-01;MIN_AGE_ALIVE;15.000000
39 Simulation_0;1;1;2018-01-01;MAX_AGE_ALIVE;49.000000
40 Simulation_0;1;1;2018-01-01;MEAN_AGE_ALIVE;23.000000
41 Simulation_0;1;1;2018-01-01;MEDIAN_AGE_ALIVE;18.000000
42 Simulation_0;1;1;2018-01-01;INFECT_RATE_ALIVE;0.080300
43 Simulation_0;1;1;2018-01-01;POP_ALIVE;20000
44 Simulation_0;1;1;2018-01-01;NUM_PARTNERS;3601
45 Simulation_0;1;1;2018-01-01;MIN_AGE_DEAD;nan
46 Simulation_0;1;1;2018-01-01;MAX_AGE_DEAD;nan
47 Simulation_0;1;1;2018-01-01;MEAN_AGE_DEAD;nan
48 Simulation_0;1;1;2018-01-01;INFECT_RATE_DEAD;-nan
49 Simulation_0;1;1;2018-01-01;POP_DEAD;0
50 Simulation_0;1;1;2018-01-01;INITIAL_INFECTIONS;1606
51 Simulation_0;1;1;2018-01-01;SIMULATION_INFECTIONS;0
52 Simulation_0;1;1;2018-01-01;INITIAL_MATCHES;3601
53 Simulation_0;1;1;2018-01-01;SIMULATION_MATCHES;0
54 Simulation_0;1;1;2018-01-01;BREAKUPS;0
55 Simulation_0;1;1;2018-01-01;TIME_TAKEN;6
56 Simulation_0;1;1;2028-01-01;MIN_AGE_ALIVE;15.000000
57 Simulation_0;1;1;2028-01-01;MAX_AGE_ALIVE;59.000000
58 Simulation_0;1;1;2028-01-01;MEAN_AGE_ALIVE;30.000000
59 Simulation_0;1;1;2028-01-01;MEDIAN_AGE_ALIVE;26.000000
60 Simulation_0;1;1;2028-01-01;INFECT_RATE_ALIVE;0.142754
61 Simulation_0;1;1;2028-01-01;POP_ALIVE;23516
62 Simulation_0;1;1;2028-01-01;NUM_PARTNERS;8432
63 Simulation_0;1;1;2028-01-01;MIN_AGE_DEAD;15.000000
64 Simulation_0;1;1;2028-01-01;MAX_AGE_DEAD;55.000000
65 Simulation_0;1;1;2028-01-01;MEAN_AGE_DEAD;31.000000
66 Simulation_0;1;1;2028-01-01;INFECT_RATE_DEAD;0.768553
67 Simulation_0;1;1;2028-01-01;POP_DEAD;795
68 Simulation_0;1;1;2028-01-01;INITIAL_INFECTIONS;1606
69 Simulation_0;1;1;2028-01-01;SIMULATION_INFECTIONS;2362
70 Simulation_0;1;1;2028-01-01;INITIAL_MATCHES;3601
71 Simulation_0;1;1;2028-01-01;SIMULATION_MATCHES;146416
72 Simulation_0;1;1;2028-01-01;BREAKUPS;141010
73 Simulation_0;1;1;2028-01-01;TIME_TAKEN;11
74 Simulation_1;2;0;2018-01-01;MIN_AGE_ALIVE;15.000000
75 Simulation_1;2;0;2018-01-01;MAX_AGE_ALIVE;49.000000
76 Simulation_1;2;0;2018-01-01;MEAN_AGE_ALIVE;23.000000

```

(continues on next page)

(continued from previous page)

```

77 Simulation_1;2;0;2018-01-01;MEDIAN_AGE_ALIVE;18.000000
78 Simulation_1;2;0;2018-01-01;INFECT_RATE_ALIVE;0.080600
79 Simulation_1;2;0;2018-01-01;POP_ALIVE;20000
80 Simulation_1;2;0;2018-01-01;NUM_PARTNERS;3569
81 Simulation_1;2;0;2018-01-01;MIN_AGE_DEAD;nan
82 Simulation_1;2;0;2018-01-01;MAX_AGE_DEAD;nan
83 Simulation_1;2;0;2018-01-01;MEAN_AGE_DEAD;nan
84 Simulation_1;2;0;2018-01-01;INFECT_RATE_DEAD;-nan
85 Simulation_1;2;0;2018-01-01;POP_DEAD;0
86 Simulation_1;2;0;2018-01-01;INITIAL_INFECTIONS;1612
87 Simulation_1;2;0;2018-01-01;SIMULATION_INFECTIONS;0
88 Simulation_1;2;0;2018-01-01;INITIAL_MATCHES;3569
89 Simulation_1;2;0;2018-01-01;SIMULATION_MATCHES;0
90 Simulation_1;2;0;2018-01-01;BREAKUPS;0
91 Simulation_1;2;0;2018-01-01;TIME_TAKEN;5
92 Simulation_1;2;0;2028-01-01;MIN_AGE_ALIVE;15.000000
93 Simulation_1;2;0;2028-01-01;MAX_AGE_ALIVE;59.000000
94 Simulation_1;2;0;2028-01-01;MEAN_AGE_ALIVE;30.000000
95 Simulation_1;2;0;2028-01-01;MEDIAN_AGE_ALIVE;26.000000
96 Simulation_1;2;0;2028-01-01;INFECT_RATE_ALIVE;0.136983
97 Simulation_1;2;0;2028-01-01;POP_ALIVE;23638
98 Simulation_1;2;0;2028-01-01;NUM_PARTNERS;8423
99 Simulation_1;2;0;2028-01-01;MIN_AGE_DEAD;15.000000
100 Simulation_1;2;0;2028-01-01;MAX_AGE_DEAD;59.000000
101 Simulation_1;2;0;2028-01-01;MEAN_AGE_DEAD;32.000000
102 Simulation_1;2;0;2028-01-01;INFECT_RATE_DEAD;0.717507
103 Simulation_1;2;0;2028-01-01;POP_DEAD;754
104 Simulation_1;2;0;2028-01-01;INITIAL_INFECTIONS;1612
105 Simulation_1;2;0;2028-01-01;SIMULATION_INFECTIONS;2167
106 Simulation_1;2;0;2028-01-01;INITIAL_MATCHES;3569
107 Simulation_1;2;0;2028-01-01;SIMULATION_MATCHES;145976
108 Simulation_1;2;0;2028-01-01;BREAKUPS;140568
109 Simulation_1;2;0;2028-01-01;TIME_TAKEN;10
110 Simulation_1;3;1;2018-01-01;MIN_AGE_ALIVE;15.000000
111 Simulation_1;3;1;2018-01-01;MAX_AGE_ALIVE;49.000000
112 Simulation_1;3;1;2018-01-01;MEAN_AGE_ALIVE;23.000000
113 Simulation_1;3;1;2018-01-01;MEDIAN_AGE_ALIVE;18.000000
114 Simulation_1;3;1;2018-01-01;INFECT_RATE_ALIVE;0.083850
115 Simulation_1;3;1;2018-01-01;POP_ALIVE;20000
116 Simulation_1;3;1;2018-01-01;NUM_PARTNERS;3550
117 Simulation_1;3;1;2018-01-01;MIN_AGE_DEAD;nan
118 Simulation_1;3;1;2018-01-01;MAX_AGE_DEAD;nan
119 Simulation_1;3;1;2018-01-01;MEAN_AGE_DEAD;nan
120 Simulation_1;3;1;2018-01-01;INFECT_RATE_DEAD;-nan
121 Simulation_1;3;1;2018-01-01;POP_DEAD;0
122 Simulation_1;3;1;2018-01-01;INITIAL_INFECTIONS;1677
123 Simulation_1;3;1;2018-01-01;SIMULATION_INFECTIONS;0
124 Simulation_1;3;1;2018-01-01;INITIAL_MATCHES;3550
125 Simulation_1;3;1;2018-01-01;SIMULATION_MATCHES;0
126 Simulation_1;3;1;2018-01-01;BREAKUPS;0
127 Simulation_1;3;1;2018-01-01;TIME_TAKEN;5
128 Simulation_1;3;1;2028-01-01;MIN_AGE_ALIVE;15.000000
129 Simulation_1;3;1;2028-01-01;MAX_AGE_ALIVE;59.000000
130 Simulation_1;3;1;2028-01-01;MEAN_AGE_ALIVE;30.000000
131 Simulation_1;3;1;2028-01-01;MEDIAN_AGE_ALIVE;26.000000
132 Simulation_1;3;1;2028-01-01;INFECT_RATE_ALIVE;0.139352
133 Simulation_1;3;1;2028-01-01;POP_ALIVE;23595

```

(continues on next page)

(continued from previous page)

```

134 Simulation_1;3;1;2028-01-01;NUM_PARTNERS;8456
135 Simulation_1;3;1;2028-01-01;MIN_AGE_DEAD;15.000000
136 Simulation_1;3;1;2028-01-01;MAX_AGE_DEAD;58.000000
137 Simulation_1;3;1;2028-01-01;MEAN_AGE_DEAD;31.000000
138 Simulation_1;3;1;2028-01-01;INFECT_RATE_DEAD;0.728205
139 Simulation_1;3;1;2028-01-01;POP_DEAD;780
140 Simulation_1;3;1;2028-01-01;INITIAL_INFECTIONS;1677
141 Simulation_1;3;1;2028-01-01;SIMULATION_INFECTIONS;2179
142 Simulation_1;3;1;2028-01-01;INITIAL_MATCHES;3550
143 Simulation_1;3;1;2028-01-01;SIMULATION_MATCHES;146061
144 Simulation_1;3;1;2028-01-01;BREAKUPS;140609
145 Simulation_1;3;1;2028-01-01;TIME_TAKEN;10

```

- The first column (name) is the simulation group name as specified in the configuration file.
- The second column (sim) is the unique simulation number. In every execution of FastSTI, each simulation will have its own unique id, starting from 0.
- The third column (num) is the number of the simulation within the simulation group. Here each of the two simulation groups (Simulation\_0 and Simulation\_1) executed two simulations each, numbered 0 and 1.
- The fourth column is the date within the simulation that this snapshot report is for. In these simulations, the start date was 1 January 2018, and the end date was 1 January 2028. The *\_report* event ran before and after the simulation (it can run as often as you like during the simulation as well).
- The fifth column is the description of what's being reported. These are described below.
- The sixth column is the value of what's being reported. E.g. 10 for TIME\_TAKEN means the simulation ran in 10 seconds up to that point.

**Warning:** The output in the example above is all in chronological order. This is because it was a single-threaded sequential execution. Normally you will do a multithreaded execution and the report lines will likely **not** come out in order. When analysing them in a spreadsheet, R or Python, remember to sort the output by the first four columns.

### 8.1.1 What the descriptions mean

**MIN\_AGE\_ALIVE** is the youngest age, in years, of a currently living agent

**MAX\_AGE\_ALIVE** is the oldest age, in years of a currently living agent

**MEAN\_AGE\_ALIVE** is the mean age, in years, of the living population

**MEDIAN\_AGE\_ALIVE** is the median age, in years, of the living population

**INFECT\_RATE\_ALIVE** is the infection rate of the living population

**POP\_ALIVE** is the number of living agents

**NUM\_PARTNERS** is the number of agents in relationships

**MIN\_AGE\_DEAD** is the age of the youngest agent who has died

**MAX\_AGE\_DEAD** is the age of the oldest agent who has died

**INFECT\_RATE\_DEAD** is the infection rate in the dead population

**POP\_DEAD** is the number of dead agents

**INITIAL\_INFECTIONS** is the number of infected agents at the start of the simulation

**SIMULATION\_INFECTIONS** is the number of agents who became infected during the simulation

**INITIAL\_MATCHES** is the number of pairs of agents in relationships at the start of the simulation

**SIMULATION\_MATCHES** is the number of pairs of agents made during the simulation

**BREAKUPS** is the number of pairs that broke up during the simulation

**TIME\_TAKEN** is the time taken by the simulation in seconds

## 8.2 Agent output

The `_write_agents_csv_header` prints out the header line in an output agent csv file. The `_write_agents_csv` event actually writes out all the agents (living and dead), and the `_write_living_agents_csv` and `_write_dead_agents_csv` write the living and dead agents respectively.

Here is an extract from sample output of the `_write_agents_csv_header` and `_write_agents_csv` events.

```

1  sim;date;id;age;sex;sex_preferred;infected;treated;resistant;date_death;partner;
   ↪change_date
2  1;2028-01-01;1547;25;0;1;0;0;0;0000-00-00;515;2035-10-24
3  1;2028-01-01;1548;41;0;1;4;0;0;2023-10-11;-1;2025-10-30
4  1;2028-01-01;1549;28;0;1;0;0;0;0000-00-00;6620;2028-04-22
5  1;2028-01-01;1550;25;1;0;0;0;0;0000-00-00;3910;2029-09-04
6  1;2028-01-01;1551;31;1;0;1;1;0;0000-00-00;22910;2028-01-13
7  1;2028-01-01;1552;30;1;0;0;0;0;0000-00-00;18055;2028-01-03
8  1;2028-01-01;1553;36;1;0;0;0;0;0000-00-00;7448;2028-10-14
9  1;2028-01-01;1554;33;1;0;0;0;0;0000-00-00;694;2028-04-25
10 1;2028-01-01;1555;28;0;0;0;0;0;0000-00-00;984;2028-03-23
11 1;2028-01-01;1556;28;0;1;0;0;0;0000-00-00;4467;2030-03-27
12 1;2028-01-01;1557;45;0;1;0;0;0;0000-00-00;15614;2035-01-22
13 1;2028-01-01;1558;27;1;0;0;0;0;0000-00-00;3367;2030-02-17
14 1;2028-01-01;1559;29;0;1;0;0;0;0000-00-00;10884;2030-01-10
15 1;2028-01-01;1560;52;1;0;0;0;0;0000-00-00;19043;2028-05-24
16 1;2028-01-01;1561;58;1;0;0;0;0;0000-00-00;13638;2028-07-01
17 1;2028-01-01;1562;30;1;0;0;0;0;0000-00-00;14842;2030-02-19
18 1;2028-01-01;1563;57;0;1;0;0;0;0000-00-00;17535;2029-03-13
19 1;2028-01-01;1564;37;1;0;1;1;1;0000-00-00;15586;2031-10-26
20 1;2028-01-01;1565;29;1;0;0;4;0;2026-11-30;-1;2027-08-30
21 1;2028-01-01;1566;46;1;0;0;0;0;0000-00-00;-1;2030-03-21
22 1;2028-01-01;1567;26;1;0;0;0;0;0000-00-00;16097;2029-03-04

```

The first column is the simulation number. The second column is the current simulation date. The third column is the unique agent id. The fourth to ninth columns are the agent's sex, preferred sex, infection stage, treatment regimen, and resistance status respectively. The tenth column is 0000-00-00 if the agent is alive, else the date the agent died. The eleventh column is the unique id of the agent's partner (-1 if the agent is single). The last column is the date the agent's relationship status will change, either to single if it's in a relationship, or to a relationship if it's single.

---

**Note:** FastSTI's agent output event assumes an agent has at most one partner. You can extend it if you're implementing concurrency.

---



---

**Note:** If you feel comfortable writing C code, you can modify or extend the output of the `_write_agents_csv`, `_write_living_agents_csv` and `_write_dead_agents_csv` events by redefining the

---

FSTI\_AGENT\_PRINT\_CSV\_HEADER and FSTI\_AGENT\_PRINT\_CSV macros. Put your redefinition in *src/fsti-userdefs.h*. The default implementation is in *src/fsti-defaults.h*.

---

---

## Configuration file parameters

---

To see the list of configuration file parameters, along with a description and the value for each, run:

```
faststi -p
```

This prints out the parameters in a logical order. Here is an alphabetically arranged reference guide to the parameters:

### 9.1 after\_events

These are the events executed after a simulation starts.

Default value: `_no_op`

Events used in: None

Examples::

```
after_events = _write_agents_csv;_report  
after_events = _no_op # No events are executed after the simulation (default)
```

### 9.2 age\_alpha

If instead of reading in an agent file, the agents are generated with the `_generate_agents` event, then the initial age of each agent is drawn a random alpha-beta distribution. This is the alpha parameter.

Default value: 0.3

Events used in: `_generate_agents`, `_generate_and_pair`

Example:

```
age_alpha = 0.5
```

## 9.3 age\_beta

If instead of reading in an agent file, the agents are generated with the `_generate_agents` event, then the initial age of each agent is drawn a random alpha-beta distribution. This is the beta parameter.

Default: 1.0

Events used in: `_generate_agents`, `_generate_and_pair`

## 9.4 age\_input\_time\_step

Denomination of agent ages used in input files. This is typically either a year or 5-year age groups.

Default: 1 year (525949 minutes)

Events used in: `_read_agents`

Examples:

```
age_input_time_step = 525949
age_input_time_step = YEAR
age_input_time_step = 5 YEARS
```

## 9.5 age\_max

Oldest age that a generated agent at the beginning of a simulation can be initialized to.

Default: 50 years (26297450 minutes)

Events used in: `_generate_agents`, `_generate_and_pair`

Examples:

```
age_max = 26297450
age_max = 50 YEARS
```

## 9.6 age\_min

Youngest age that a generated agent at the beginning of a simulation can be initialized to.

Default: 15 years (7889235 minutes)

Events used in: `_generate_agents`, `_generate_and_pair`

Examples:

```
age_max = 26297450
age_max = 50 YEARS
```

## 9.7 agents\_input\_file

Name of the csv file to read agents from. Use empty string for standard input.

Default: agents\_in.csv

Events used in: \_read\_agents

Examples:

```
agents_input_file = agents_in.csv
agents_input_file = stdin # Reads agents from standard input
```

## 9.8 agents\_output\_file

Name of the csv file to write agents to. Use empty string for standard output.

Events used in: \_write\_agents\_csv\_header, \_write\_agents\_csv, \_write\_living\_agents\_csv, \_write\_dead\_agents\_csv

Examples:

```
agents_output_file = agents_out.csv
agents_output_file = stdout # Writes agents to standard output
```

## 9.9 before\_events

These are the events executed before a simulation starts.

Default: \_no\_op

Events used in: None

Examples:

```
before_events = _write_agents_csv_header;_write_results_csv_header;_generate_and_pair
before_events = _no_op # No events are executed after the simulation (default)
```

## 9.10 birth\_event\_every\_n

Indicates how frequently (i.e. every nth iteration) the birth (\_birth) event is executed.

The birth event creates new agents during the simulation. Each new agent is set to the minimum age. Because the number of agents might be too small for their to be births on every iteration, this parameter allows you to execute the births at specified iterations.

Also, births are discrete. There can only be a whole number of births executed by the `_birth` event. Even if births can take place daily, rounding the expected number of births to a whole number may create a severely inaccurate birth rate. Therefore it may be better to execute the `_birth` event infrequently but with more births taking place each time it executes, in order to reduce the difference between the births generated by the model and the births in the real-world population being studied.

Default: 73 (i.e. every 73rd iteration - by default this would be every 73rd day of the simulation)

Of course if your simulation doesn't use the `_birth` event, this parameter is irrelevant.

Events used in: `_birth`

Examples:

```
birth_event_every_n = 73
```

## 9.11 birth\_rate

Birth rate for the period of `birth_event_every_n`

Events used in: `_birth`

Default: 0.003968

Example:

```
birth_rate = 0.002
```

## 9.12 csv\_delimiter

Character that separates CSV fields

Events used in: any event that reads or writes a CSV file.

Default value: `;`

Examples:

```
csv_delimiter = ,
```

---

**Note:** It's not currently possible to set the `csv_delimiter` in the configuration file to a semi-colon (`;`) because this is the delimiter for the configuration file itself. But the default value for `csv_delimiter` is the semi-colon, so it should be unnecessary to have to set it to anything other than a comma.

---

## 9.13 dataset\_birth\_infect

Specifies the location of a dataset used to set the infection stage of agents, if any, when they enter the simulation (when agents are born, so to speak, although since they are born at the minimum age of the simulation, e.g. 15 years old, they may already be sexually active).

See the `data/dataset_birth_infect.csv` file for an example of this dataset.

In this example file, the the agent characteristics of sex, sex\_preferred and age (in 10-year groupings) are used to determine probability of an agent being uninfected (i.e. agent->infected is set to 0), or stage 1, 2, 3 or 4. Note the probabilities are ascending from stage 1 through 4. The \_birth event first checks if a uniform random number, r, is < than the stage 1 probability. If it is, agent->infected is set to 1. Then it checks if r is < than the stage 2 probability and >= the stage 1 probability. If it is, agent->infected is set to 2. Etc. If r is >= the stage 4 probability, the agent is uninfected and agent->infected is set to 0. Your simulation can have many stages (up to 254, but this would almost certainly be unmanageable), so long as they are consistently treated across datasets.

Events used in: \_birth

Default: \_no\_op # i.e. there is no dataset file specified.

Examples:

```
dataset_birth_infect = dataset_birth_infect.csv
```

## 9.14 dataset\_birth\_resistant

Specifies the location of a dataset used to set the resistance of infected agents, if any, when they enter the simulation (when agents are born, so to speak, although since they are born at the minimum age of the simulation, e.g. 15 years old, they may already be sexually active). The file can have any number of columns specifying agent characteristics (independent variables). It must have exactly one dependent variable column specifying the risk of resistance for agents with a given set of characteristics.

See the data/dataset\_birth\_resistant.csv file for an example of this dataset.

Events used in: \_birth

Default: \_no\_op # i.e. there is no dataset file specified.

Examples:

```
dataset_birth_resistant = dataset_birth_resistant.csv
```

---

**Note:** This mechanism for modelling resistance at birth is a bit too simple and needs to be improved.

---

## 9.15 dataset\_birth\_treated

Specifies the location of a dataset used to set the probability of an infected agent being on treatment when they enter the simulation (when agents are born, so to speak, although since they are born at the minimum age of the simulation, e.g. 15 years old, they may already be sexually active). The file can have any number of columns specifying agent characteristics (independent variables). It must have exactly one dependent variable column specifying the probability of treatment for agents with a given set of characteristics

See the data/dataset\_birth\_treated.csv file for an example of this dataset.

Events used in: \_birth

Default: \_no\_op # i.e. there is no dataset file specified.

Examples:

```
dataset_birth_treated = dataset_birth_treated.csv
```

## 9.16 dataset\_coinfect

Specifies the location of a dataset used to set the coinfection status of an agent. You can have as many columns specifying agent characteristics (i.e. the independent variables) as you wish but the `_coinfect` event expects exactly one dependent variable, the probability of the agent being coinfecting per time step.

See the `data/dataset_coinfect.csv` file for an example of this dataset.

Events used in: `_coinfect`

Examples:

```
dataset_coinfect = dataset_coinfect.csv
```

## 9.17 dataset\_gen\_infect

Specifies the location of a dataset used to set the infection stage of agents, if any, at the beginning of a simulation.

See the `data/dataset_gen_infect.csv` file for an example of this dataset.

In this example file, the agent characteristics of `sex`, `sex_preferred` and `age` (in 10-year groupings) are used to determine probability of an agent being uninfected (i.e. `agent->infected` is set to 0), or stage 1, 2, 3 or 4. Note the probabilities are ascending from stage 1 through 4. The `_birth` event first checks if a uniform random number, `r`, is `<` than the stage 1 probability. If it is, `agent->infected` is set to 1. Then it checks if `r` is `<` than the stage 2 probability and `>=` the stage 1 probability. If it is, `agent->infected` is set to 2. Etc. If `r` is `>=` the stage 4 probability, the agent is uninfected and `agent->infected` is set to 0. Your simulation can have many stages (up to 254, but this would almost certainly be unmanageable), so long as they are consistently treated across datasets.

Events used in: `_generate_agents`, `_generate_and_pair`

Default: `_no_op` # i.e. there is no dataset file specified.

Examples:

```
dataset_gen_infect = dataset_gen_infect.csv
```

## 9.18 dataset\_gen\_mating

Specifies the location of a dataset used to set the probability of an agent being in the mating pool at the beginning of a simulation. You can have as many columns specifying agent characteristics (i.e. the independent variables) as you wish but the events that use this dataset expect exactly one dependent variable, the probability of the agent being in the initial mating pool.

See the `data/dataset_gen_mating.csv` file for an example of this dataset.

Events used in: `_generate_agents`, `_generate_and_pair`

Default: `_no_op` # i.e. there is no dataset file specified.

Examples:

```
dataset_gen_mating = dataset_gen_mating.csv
```

## 9.19 dataset\_gen\_resistant

Specifies the location of a dataset used to set the resistance of infected agents, if any, at the beginning of a simulation. The file can have any number of columns specifying agent characteristics (independent variables). It must have exactly one dependent variable column specifying the risk of resistance for agents with a given set of characteristics.

See the data/dataset\_gen\_resistant.csv file for an example of this dataset.

Events used in: \_generate\_agents, \_generate\_and\_pair

Default: \_no\_op # i.e. there is no dataset file specified.

Examples:

```
dataset_gen_resistant = dataset_gen_resistant.csv
```

---

**Note:** This mechanism for modelling resistance is a bit too simple and needs to be improved.

---

## 9.20 dataset\_gen\_sex

Specifies the location of a dataset used to set the sex of an agent at the beginning of a simulation. The file can have zero or more columns specifying agent characteristics (independent variables). It must have exactly one dependent variable column specifying the probability of the agent being male. Typically this is a one-column dataset with a header and one data row set to 0.5. But if you want need more sophisticated initiation of agent sex (e.g. by age), then this is the dataset in which you specify it.

See the data/dataset\_gen\_sex.csv file for an example of this dataset.

Events used in: \_generate\_agents, \_generate\_and\_pair

Default: \_no\_op # i.e. there is no dataset file specified.

Examples:

```
dataset_gen_sex = dataset_gen_sex.csv
```

## 9.21 dataset\_gen\_sex\_preferred

Specifies the location of a dataset used to set the sexual preference of an agent at the beginning of a simulation. The file can have zero or more columns specifying agent characteristics (independent variables). It must have exactly one dependent variable column specifying the probability of the agent preferring a male sexual partner.

See the data/dataset\_gen\_sex\_preferred.csv file for an example of this dataset.

Default: \_no\_op # i.e. there is no dataset file specified.

Examples:

```
dataset_gen_sex_preferred = dataset_gen_sex_preferred.csv
```

## 9.22 dataset\_gen\_treated

Specifies the location of a dataset used to set the treatment status of an infected agent at the beginning of a simulation. The file can have zero or more columns specifying agent characteristics (independent variables). The number of dependent variable columns must correspond to the number of possible treatment statuses, incrementing from 1. Events that use this dataset generate a uniform random number,  $r$ , and then compare  $r$  from the first dependent column onwards. If  $r$  is less than the probability in a dependent column, the agent's treatment status is set to the dependent column number.

Here's a mixture of C and pseudocode showing how FastSTI does this:

```
1  num_stages = simulation->dataset_gen_treated->num_dependents;
2  rnd = uniform random number;
3  agent->treated = 0;
4  row = fsti_dataset_lookup_row(dataset_gen_treated, agent);
5  for (col = 1; col <= num_stages; col++) {
6      prob = dataset_get(dataset_gen_treated, row, col);
7      if (rnd < prob) {
8          agent->treated = col;
9          break;
10     }
11 }
```

See the data/dataset\_gen\_treated.csv file for an example of this dataset.

Default: `_no_op` # i.e. there is no dataset file specified.

Examples:

```
dataset_gen_treated = dataset_gen_sex_treated.csv
```

## 9.23 dataset\_infect

Specifies the location of a dataset used to determine whether an agent becomes infected by its sexual partner. This is a two-agent dataset, since the probability of infection is a function of the characteristics of both agents. See [Two-agent datasets](#) for details on how this works.

Default: `_no_op` # i.e. there is no dataset file specified.

Examples:

```
dataset_gen_infect = dataset_gen_infect.csv
```

## 9.24 dataset\_infect\_stage

Specifies the location of a dataset used to determine if an infected agent should change the infection stage it is in.

Default: `_no_op` # i.e. there is no dataset file specified.

Examples:

```
dataset_gen_infect = dataset_gen_infect.csv
```

This is quite a complicated dataset and is best understood by looking at the commented example in the data directory called `dataset_infect_stage.csv`. For convenience here it is:

```

1  # Dataset used by _stage event (defined in fsti-events.c as fsti_event_stage)
2  #
3  # The first three columns are used for matching and correspond to agent fields
4  # or properties.
5  #
6  # The next six columns are instructions on how and when to change the stage.
7  #
8  # Columns:
9  #
10 # 1. infected - the infection stage of the agent (0 is uninfected)
11 # 2. treated - the treatment regimen of the agent. This particular file
12 #    allows for 3 treatment regimens.
13 # 3. resistant - 0 if the agent is drug-susceptible to this treatment regimen
14 #    1 if the agent is drug-resistant to this treatment regimen
15 # 4. prob_stage_change - probability that the infection stage changes for
16 #    this time step (1 day)
17 # 5. stage_incr - if a uniformly generated random number < prob_stage_change
18 #    then change the infect property by this increment
19 # 6. prob_treatment_change - probability treatment changes
20 # 7. treatment_incr - if a uniformly rand number < prob_treatment_change
21 #    then change the treatment property by this increment
22 # 8. prob_resistant - probability resistance status changes
23 # 9. resistant_incr - if a uniformly random number < prob_resistant_change
24 #    change the resistant value by this amount
25 #
26 # Note the |6 after resistant_incr means that there are six columns at the end
27 # of each line that are not agent properties.
28 #
29 # Infection stages:;;;;;;;;;
30 # 0 = uninfected;;;;;;;;;
31 # 1 = virally suppressed (usually on treatment);;;;;;;;;;
32 # 2 = primary infection (highly infectious);;;;;;;;;;
33 # 3 = chronic infection;;;;;;;;;
34 # 4 = Final stage;;;;;;;;;
35 #
36 # HEADER ROW FOLLOWS
37 infected;treated;resistant;prob_stage_change;stage_incr;prob_treatment_change;
→treatment_incr;prob_resistant;resistant_incr|6
38 0;0;0;0;0;0;0;0;0;0;0
39 0;0;1;0;0;0;0;0;0;0;0
40 0;1;0;0;0;0;0;0;0;0;0
41 0;1;1;0;0;0;0;0;0;0;0
42 0;2;0;0;0;0;0;0;0;0;0
43 0;2;1;0;0;0;0;0;0;0;0
44 0;3;0;0;0;0;0;0;0;0;0
45 0;3;1;0;0;0;0;0;0;0;0
46 1;0;0;0.1;1;0.0001;1;0;0;0;0
47 1;0;1;0.1;1;0.0001;1;0;0;0;0
48 1;1;0;0.00001;1;0.00001;1;0.00001;1;0;0;0
49 1;1;1;0.1;1;0.0001;1;0;0;0;0
50 1;2;0;0.00001;1;0.00001;1;0.00001;1;0;0;0
51 1;2;1;0.1;1;0.0001;1;0;0;0;0
52 1;3;0;0.00001;1;0;0;0.0001;1;0;0;0
53 1;3;1;0.1;1;0;0;0;0;0;0
54 2;0;0;0.1;1;0.001;1;0;0;0;0
55 2;0;1;0.1;1;0.001;1;0;0;0;0
56 2;1;0;0.1;-1;0;0;0.0001;1;0;0;0

```

(continues on next page)

(continued from previous page)

```

57 2;1;1;0.1;1;0.001;1;0;0
58 2;2;0;0.1;-1;0;0;0.0001;1
59 2;2;1;0.1;1;0.001;1;0;0
60 2;3;0;0.1;-1;0;0;0.0001;1
61 2;3;1;0.1;1;0;0;0;0
62 3;0;0;0.004;1;0.0001;1;0;0
63 3;0;1;0.001;1;0.0001;1;0;0
64 3;1;0;0.1;-1;0;0;0.0001;1
65 3;1;1;0.002;1;0.005;1;0;0
66 3;2;0;0.1;-1;0;0;0.0001;1
67 3;2;1;0.002;1;0.001;1;0;0
68 3;3;0;0.1;-1;0;0;0.0001;1
69 3;3;1;0.002;1;0;0;0;0
70 4;0;0;0;0;0.005;1;0;0
71 4;0;1;0;0;0.005;1;0;0
72 4;1;0;0.1;-1;0;0;0.0001;1
73 4;1;1;0;0;0.01;1;0;0
74 4;2;0;0.1;-1;0;0;0.0001;1
75 4;2;1;0;0;0.01;1;0;0
76 4;3;0;0.1;-1;0;0;0.0001;1
77 4;3;1;0;0;0;0;0;0

```

## 9.25 dataset\_mortality

Specifies the location of a dataset used to determine if an agent should die on the current time step.

See the data/dataset\_mortality.csv file for an example of this dataset.

Default: `_no_op` # i.e. there is no dataset file specified.

Examples:

```
dataset_mortality = dataset_mortality.csv
```

## 9.26 dataset\_rel\_period

Specifies the location of a dataset used to determine the length of time an agent relationship should be.

This dataset requires two dependent variable columns: `scale` and `shape`, which are used to draw a random number from a Weibull distribution for each agent in the relationship. The values returned are the number of time steps (iterations). The duration of the relationship is the mean of the two random numbers drawn.

Default: `_no_op` # i.e. there is no dataset file specified.

Examples:

```
dataset_rel_period = dataset_rel_period.csv
```

## 9.27 dataset\_single\_period

Specifies the location of a dataset used to determine the length of time an agent should remain single.

This dataset requires two dependent variable columns: scale and shape, which are used to draw a random number from a Weibull distribution for the agent. The value drawn is the number of time steps the agent should remain single.

Default: `_no_op` # i.e. there is no dataset file specified.

Examples:

```
dataset_single_period = dataset_single_period.csv
```

## 9.28 during\_events

These are the events executed on every time step of a simulation.

Default: `_no_op`

Events used in: None

Examples:

```
during_events=_age;_breakup_and_pair;_infect;_stage;_birth;_death
during_events = _no_op # No events are executed during the simulation (default)
```

## 9.29 report\_frequency

Indicates the frequency that the reporting events specified by the `during_events` parameter should be specified.

Default: 1, i.e. on every iteration or time step of the simulation. Make sure this is what you really want. If you're executing `_write_agents_csv` during the simulation, having a `report_frequency` of 1 will slow your simulation down and use huge amounts of hard drive space.

Events used in: `_write_agents_csv`, `_write_live_agents_csv`, `_write_dead_agents_csv`, `_report`

Examples:

```
report_frequency = 365 # Report every 365 time steps.
```

## 9.30 initial\_infect\_stage

When an agent is infected, this parameter determines the initial infection stage.

Default: 2. In the default simulation provided by FastSTI, the agent infected property of 1 implies it is on treatment. No one is on treatment when infected, so 2 represents the primary infection stage, and this is therefore the default value that `initial_infect_stage` is set to.

Events used in: `_infect`

Example:

```
initial_infect_stage = 2
```

## 9.31 match\_k

The value of k when using nearest-neighbour type agent matching algorithms. The agent matching algorithm supplied with GroundUp (implemented by the `_rkpm` event) chooses the best match for the current agent being considered for matching in the mating pool from the k unmatched agents adjacent to it.

If k is set to 1, this is random matching. If k is set larger than the possible number of agents in the mating pool, then this is a kind of brute force matching.

Default: 100 This is usually a good compromise value. The algorithm will execute quickly and the agent selected will on average be a better match than 99% of the remaining agents in the mating pool.

Default: 100

Events used in: `_rpkm`

Example:

```
match_k = 300
```

## 9.32 mutual\_csv\_partners

Sometimes a CSV file of agents that is used to initialize a simulation only records one of the partners in a relationship. If this is the case then set this parameter to 1. FastSTI will then iterate through all the agents after they've been read in, but before the simulation loop begins, and properly initialise all relationships.

Leaving this at 0 if the CSV file does not mutually record agent relationships will give unpredictable results. But setting to 1 is always harmless, albeit resulting in very slightly slower execution.

This parameter has no effect if agents are generated (i.e. not read in via a CSV file).

Default: 1

Events used in: `_read_agents`

Example:

```
mutual_csv_partners = 1
```

## 9.33 num\_agents

Determines the number of agents to generate for a simulation.

This parameter has no effect if agents are read in from a CSV file.

Default: 20000

Events used in: `_generate_agents`, `_generate_and_pair`

Example:

```
num_agents = 10000
```

## 9.34 num\_simulations

Indicates the number of times to repeat a simulation in a simulation group.

Default: 1

Events used in: None

Example:

```
num_simulations = 1
```

## 9.35 partnerships\_file

The csv file name to output partnerships. Set to empty string for stdout. This file will be empty unless one or more of record\_breakups, record\_infections or record\_matches are set.

Default: Empty string (i.e. stdout)

Events used in: \_rkpm, \_infect, \_breakup, \_breakup\_and\_pair, \_generate, \_generate\_and\_pair

Example:

```
partnerships_file = "my_partnerships.csv"
```

TO DO

prob\_birth\_infected\_msm; Probability a new msm agent is infected; 0.001000

prob\_birth\_infected\_msw; Probability a new msw agent is infected; 0.000100

prob\_birth\_infected\_wsm; Probability a new wsm agent is infected; 0.000500

prob\_birth\_infected\_wsw; Probability a new wsw agent is infected; 0.000100

prob\_birth\_male; Probability a new agent is male; 0.500000

prob\_birth\_msw; Probability a new male agent is msw; 0.950000

prob\_birth\_wsm; Probability a new female agent is wsm; 0.950000

prob\_gen\_male; Probability a generated agent is male; 0.500000

prob\_gen\_msw; Probability a generated male agent is msw; 0.950000

prob\_gen\_wsm; Probability a generated female agent is wsm; 0.950000

record\_breakups; Whether to output breakups to the partnership file; 0

record\_infections; Whether to output infections to the partnership file; 0

record\_matches; Whether to output matches to the partnership file; 0

results\_file; File name to output results to (empty string for stdout);

simulation\_period; Time period of the simulation (10 years); 5259490

stabilization\_events; Events used to stabilize the agent characteristics before the actual simulation; \_no\_op

stabilization\_steps; Number of time steps to run before executing various events; 0

start\_date; Start date of simulation (yyyy;mm;dd); 2018; 1; 1

threads; Number of threads (0=system determined); 0

time\_step; Time step for each iteration of simulation in minutes(default 1440 minutes == 1 day); 1440  
treatment\_infect\_stage; When treated this is the integer to set infected to; 1

# CHAPTER 10

---

## Events

---

The events to run for a simulation are specified in these parameters: - before\_events, - during\_events, and - after\_events.

These, respectively, are the events run once at the start of each simulation (before\_events), on every iteration of the simulation (during\_events), and once at the end of the simulation (after\_events).

Here is a way to specify a comprehensive simulation:

```
1 before_events = _write_agents_csv_header; _write_results_csv_header; _generate_and_
  ↳ pair; _write_agents_csv; _report
2
3 during_events = _age; _breakup_and_pair; _infect; _stage; _birth; _death; _report
4
5 after_events = _write_agents_csv
```

This tells FastSTI that before each simulation is run, it must:

- Write the header line for the output agents csv file (\_write\_agents\_csv\_header)
- Write the header line for the output results file (\_write\_results\_csv\_header)
- Generate and initialize properly a new set of agents and put some of them into relationships (\_generate\_and\_pair)
- Write the set of initialised agents to a CSV file.
- Write some stats (e.g. prevalence, agents alive, agents dead etc) to the results file (\_report).

On every time step of a simulation, it tells FastSTI to:

- Increment each living agent's by the time step (\_age).
- Iterate through the living agents and put some of the single ones into relationships and break up some of those that are in relationships (\_breakup\_and\_pair).
- Iterate through the agents with partners and infect some of those in sero-discordant relationships (\_infect).
- Iterate through the infected agents and move some of them to a new disease stage (e.g. treated, resistant, ill - whatever you specify actually).

- Add some new agents to the simulation at the minimum age (`_birth`).
- Kill some of the agents (`_death`).
- Write some stats (e.g. prevalence, agents alive, agents dead etc) to the results file (`_report`).

At the end of each simulation it tells FastSTI to:

- Write the final state of the agents to a CSV file (`_write_agents_csv`).

The events provided by FastSTI (which include those in the example above) are prefixed with an underscore (`_`), to differentiate them from other 3rd-party events or ones that you choose to implement. Please don't name your events with a leading underscore.

If the events provided by FastSTI are not all you need, then you are encouraged to code your own events in C in the source code files `fsti-userdefs.h` and `fsti-userdefs.c`.

## 10.1 `_read_agents`

Reads in a csv file of agents at the beginning of a simulation. The file name is given by the `agents_input_file` parameter. The csv file delimiter is given by the `csv_delimiter` parameter, which defaults to a semi-colon (`;`).

The first time this event is executed, it saves the agents in memory so that on subsequent simulations, it doesn't have to process the file again.

If you don't have a file of agents to read in, consider using the `_generate_agents` event.

Parameters: `agents_input_file`, `csv_delimiter`, `mutual_csv_partners`

Datasets: None

## 10.2 `_generate_agents`

Generates agents for a simulation instead of reading them from a file.

It uses the `num_agents` parameter to determine the number of agents to generate.

The default implementation sets the following agent properties: age, sex, sex\_preferred, infected, treated and resistant.

- age is set via a random beta distribution determined by the parameters `age_alpha` and `age_beta`.
- sex is set via the dataset `dataset_gen_sex`. (See `data/dataset_gen_sex.csv` for an example.)
- sex\_preferred is set via the dataset `dataset_gen_sex_preferred`. (See `data/dataset_gen_sex_preferred.csv` for an example, which sets the agents to prefer the opposite sex with a probability of 95%, but you can change this according to the needs of your model.)
- infected (i.e. the infection stage of the agent with 0 by convention meaning the agent isn't infected) is set via the dataset `dataset_gen_infect`. (See `data/dataset_gen_infect.csv` for an example.)
- treated (i.e. which treatment line, if any, the agent is on) is set via the dataset `dataset_gen_treated`. (See `data/dataset_gen_treated.csv` for an example.)
- resistant (i.e. which treatment regimens the agent is resistant to, if any. (See `data/dataset_gen_resistant.csv` for an example.)

If you wish to set additional agent properties, you must provide hook in by defining a macro called `FSTI_HOOK_GENERATE_AGENT` in `fsti-userdefs.h`. The macro takes two parameters: `simulation` (a pointer to the current simulation) and `agent` (a pointer to the current agent whose property you wish to set).

---

**Tip:** Instead of using this event, it will usually make more sense to use the `_generate_and_pair` event which generates the initial set of agents, places them in a mating pool, shuffles the mating pool, mates the agents in the mating pool and then sets the number of time steps each agent will stay in its current relationship or stay single. This event is in fact called by `_generate_and_pair`.

---

See also: `_generate_and_pair`

Parameters: `num_agents`, `age_alpha`, `age_beta`,

Datasets: `dataset_gen_sex`, `dataset_gen_sex_preferred`, `dataset_gen_treated`, `dataset_gen_resistant`

## 10.3 `_age`

Iterates through all the living agents, and adds the time increment of the simulation to their age.

This is one of the simplest events provided by FastSTI, and so it makes a nice example of how events are implemented. Here is the source code:

```

1  void
2  fsti_event_age(struct fsti_simulation *simulation)
3  {
4      struct fsti_agent *agent;
5      FSTI_FOR_LIVING(*simulation, agent, {
6          agent->age += simulation->time_step;
7      });
8  }
```

All events are declared like this, i.e. a void function that takes one parameter: a pointer to the simulation itself.

On line 4 we declare a pointer to an agent on line three. When we iterate through the living agents, this will be a pointer to the current agent the code acts upon.

The `FSTI_FOR_LIVING` macro implements a for loop over the living agents. The code inside the macro's curly brackets simply adds the time step to each agent's age.

parameters: `time_step`

Datasets: None

## 10.4 `_death`

Iterates through the living agents and kills some of them.

Datasets: `dataset_mortality`

Here is a simple example of this dataset:

```

1  infected;0
2  0;0.00000296
3  1;0.00000315
4  2;0.00000315
5  3;0.00000630
6  4;0.001
```

The first column tells the event to determine the infection stage of the agent. The second column is the probability of the agent dying on this time step. Here the probabilities are specified per day. If you change the time step to, say, a week you have to update the probabilities in this file accordingly.

## 10.5 `_initial_mating`

Before a simulation starts but after agents have been generated or read in from a file, it is possible that none of the agents are in sexual relationships.

This event is responsible for creating the initial mating pool of agent sexual relationships. It is typically only run once per simulation, and only if the agent input file doesn't specify relationships. It is set as an event to run in the `before_events` parameter.

Note that it doesn't actually put the agents into relationships, only into a mating pool. An agent pairing event, such as `_rkpm` must then be executed in order to actually place agents in relationships with each other.

---

**Tip:** Instead of using this event, it will usually make more sense to use the `_generate_and_pair` event which generates the initial set of agents, places them in a mating pool, shuffles the mating pool, mates the agents in the mating pool and then sets the number of time steps each agent will stay in its current relationship or stay single. This event is in fact called by `_generate_and_pair`.

---

Datasets: `dataset_gen_mating`

Here is an example of this dataset. The first column is age in five-year periods. So for example line 5 corresponds to the probability of a person aged 15 to just shy of 20 being in a relationship (which in this example is 0.3 or 30%).

```
1  age|5-YEAR;0
2  0;0.0
3  1;0.0
4  2;0.0
5  3;0.3
6  4;0.35
7  5;0.4
8  6;0.45
9  7;0.5
10 8;0.5
11 9;0.5
12 10;0.5
13 11;0.5
14 12;0.4
15 13;0.4
16 14;0.35
17 15;0.3
18 16;0.25
19 17;0.2
20 18;0.15
21 19;0.1
22 20;0.5
23 21;0
```

See also: `_generate_and_pair`

## 10.6 \_initial\_rel

For each living agent make a correction to the duration (number of time steps) its current relationship, or if the agent is single, set the period it will stay single.

This event assumes the relchange (the date/time in the future at which it's current relationship or single status changes) property of agents in relationships has been set. It multiplies it by a uniform random number between 0 and 1. If the agent is single it sets the single period and also multiplies it by a uniform random number between 0 and 1.

Why use this event? Because the simulation starts at an arbitrary time point in which people are already in the middle of relationships or a period of being being single. This event will on average halve the value of relchange. Whether that's a valid assumption at the beginning of a simulation is unclear to us.

---

**Tip:** Instead of using this event, it will usually make more sense to use the `_generate_and_pair` event which generates the initial set of agents, places them in a mating pool, shuffles the mating pool, mates the agents in the mating pool and then sets the number of time steps each agent will stay in its current relationship or stay single. This event is in fact called by `_generate_and_pair`.

---

Datasets: None

See also: `_generate_and_pair`, `_breakup` and `_rkpm`. The latter two also set the relchange property.

## 10.7 \_mating\_pool

Iterates through the living agents and places the single ones into the mating pool if they are due for a relationship status change. The relchange property of each single agent determines whether it is to be placed in the mating pool.

Note that placing agents in the mating pool is necessary but not sufficient to pair them into relationships. This event is typically followed by shuffling the mating pool (`_shuffle_mating`) and then placing them in sexual relationships with other agents in the mating pool using the pairing algorithm (`_rkpm`).

All of these events are included in the composite event `_breakup_and_pair`.

The C code for this event is simple and instructive:

```

1  void
2  fsti_event_mating_pool(struct fsti_simulation *simulation)
3  {
4      struct fsti_agent *agent;
5      fsti_agent_ind_clear(&simulation->mating_pool);
6      FSTI_FOR_LIVING(*simulation, agent, {
7          if (agent->num_partners == 0) {
8              if (agent->relchange[0] < simulation->iteration)
9                  fsti_agent_ind_push(&simulation->mating_pool, agent->id);
10         }
11     });
12 }
```

All events are declared like this, i.e. a void function that takes one parameter: a pointer to the simulation itself.

On line 4 we declare a pointer to an agent on line three. When we iterate through the living agents, this will be a pointer to the current agent the code acts upon.

The FSTI\_FOR\_LIVING macro implements a for loop over the living agents.

The code inside the macro's curly brackets first checks that the agent is single (i.e. it has zero partners). If it is it checks if the `relchange` property is less than the current iteration. If it is, it places the agent in the mating pool.

You may have noticed that `agent->relchange` on line 8 is subscripted with a 0 index. This is because FastSTI's data structures support agents having multiple concurrent partners. `relchange[0]` refers to the status of the first partner. By default, up to three partners are supported and this is determined by `FSTI_MAX_PARTNERS` set in `fsti-defaults.h`. To override this value, either with a bigger or smaller maximum number of partners, simply define an alternative value for `FSTI_MAX_PARTNERS` in `fsti-userdefs.h`. For example:

```
#define FSTI_MAX_PARTNERS 1
```

But although the FastSTI data structures support concurrent partnerships, all of the default events, including this one, do not support agents having more than one partner. This may and probably should change in the future. Of course you are also welcome to implement your own events that do account for concurrent partnerships.

Parameters: None

Datasets: None

See also: *\_breakup\_and\_pair*.

## 10.8 *\_breakup*

Iterates through the living agents, and breaks up some of those who are in relationships.

The event looks at the `relchange` properties of each agent in a relationship. If `relchange` is less than the current iteration, it's time for the relationship to end.

The period that each agent remains single is determined by the *dataset\_single* dataset.

If you wish to record all the breakups, set the `record_breakups` parameter to 1 and the `partnerships_file` parameter to the name of the file to output to. But note that the number of breakups in a large simulation can be huge.

Parameters: `record_breakups`, `partnerships_file`

Datasets: *dataset\_single*

See also: *\_breakup\_and\_pair*.

## 10.9 *\_shuffle\_living*

Shuffles the living agents array. This is useful if an event is biased by the order in which it processes the agents. If the agents are shuffled before the event is run, over the long run this may remove the bias.

Parameters: None

Datasets: None

## 10.10 *\_shuffle\_mating*

Shuffles the mating pool. This is important to run before many pair-matching algorithms including the *\_rkpm* event provided by FastSTI. Stochastic pair-matching events tend to assign better matches to agents at the beginning of an array. By shuffling the mating pool, this bias may be mitigated against over the long run.

Parameters: None

Datasets: None

See also: *\_breakup\_and\_pair*.

## 10.11 *\_rkpm*

This event runs the pair matching algorithm provided by FastSTI. For most purposes it's good and flexible.

RKPM stands for Random-k Pair-Matching. It may also have been called a k-Nearest Neighbour algorithm. It looks at the k agents adjacent (to the right if you think of an array as a set of objects arranged left to right) of the current agent in the mating pool, and selects the best match based on a distance measure.

The value of k is determined by the *match\_k* parameter. Its default value is 100.

If k is set to 1 in the configuration input file, then in effect agents in the mating pool are randomly matched with each other. This is a very fast way to match agents but will not generate realistic matches.

If k is set to a large number greater than or equal to the possible number of agents that will enter the mating pool (e.g. set it to 1,000,000,000 to ensure it's bigger than any practically conceivable mating pool), then in effect agents in the mating pool are matched using a brute force algorithm, i.e. the entire remainder of the mating pool is searched for a mate for the current agent. This usually obtains a set of matches that resemble the population being studied (assuming the distance measure is well implemented) but it can be very slow.

The distance measure is written in C. It can be easily modified by redefining the `FSTI_AGENT_DISTANCE` macro in `fsti-userdefs.h` and recompiling FastSTI.

This is how the default `FSTI_AGENT_DISTANCE` macro is defined:

```
1  #ifndef FSTI_AGENT_DISTANCE
2  #define FSTI_AGENT_DISTANCE(agent_a, agent_b) \
3      fsti_agent_default_distance(agent_a, agent_b)
4  #endif
```

The `fsti_agent_default_distance` function is defined in `fsti-agent.c` file and is very simple.

```
1  float fsti_agent_default_distance(const struct fsti_agent *a,
2                                  const struct fsti_agent *b)
3  {
4      float result = 0.0;
5      if (a->sex_preferred != b->sex)
6          result += 25.0;
7      if (b->sex_preferred != a->sex)
8          result += 25.0;
9      result += abs(a->age - b->age);
10     return result;
11 }
```

It minimises the distance between ages of compatible sexual orientation and similar age.

To define your own distance function, write a function to replace this in `fsti-userdefs.c` and redefine `FSTI_AGENT_DISTANCE` to call it.

If you wish to record all the matches in a file, set the `record_matches` parameter to 1 and the `partnerships_file` parameter to the name of the file to output to. But note that the number of partnerships in a large simulation can be huge.

The duration of the relationship is determined by the *dataset\_rel* dataset.

Parameters: `record_matches`, `partnerships_file`

Datasets: `dataset_rel`

## 10.12 `_breakup_and_pair`

This is a composite event that executes the following events in this order:

- `_breakup`
- `_mating`
- `_shuffle_mating`
- `_rkpm`

## 10.13 `_generate_and_pair`

This is a composite event that executes the following events in this order:

- `_generate_agents`
- `_initial_mating`
- `_shuffle_mating`
- `_rkpm`
- `_initial_rel`

## 10.14 `_infect`

Iterates the living agent and infects some of those in sero-discordant relationships.

Whether an agent becomes infected depends on both its own characteristics and those of its partner. See [Two-agent datasets](#) for details.

If an agent becomes infected, the initial value of its `infect` property is set to the `initial_infect_stage` parameter. The default is 2. Although the infection stages are entirely user-defined, in the default settings of FastSTI, the following default setup is assumed for the `infect` property:

- 0: The agent is uninfected (this should be the case for any simulation)
- 1: The agent is infected but on treatment
- 2: The agent is in a primary infection stage
- 3: The agent is in a chronic infection stage
- 4: The agent is sick or in an end-stage of infection

This can be specified differently. But you must make sure that your stages are consistent across the simulation, else nonsensical things will happen. If you want a different infection stage setup, make sure your datasets and `initial_infect_stage` parameter are consistent with each other.

If you wish to record all the infections, set the `record_infections` parameter to 1 and the `partnerships_file` parameter to the name of the file to output to.

Parameters: `record_infections`, `partnerships_file`

Dataset: `dataset_infect`

## 10.15 `_stage`

Iterates over the living agents and changes the infection stage of some of the infected agents. It can also change the agent's treatment and resistant properties.

How many stages there are for the infection is entirely user-defined, but you have to make sure that the stages are consistent across events. Also, the possible values for the treatment and resistant agent properties are also user-defined, but we think the default values FastSTI has been set up with are sensible for many models.

Although the infection stages are entirely user-defined, in the default settings of FastSTI, the following default setup is assumed for the infect property:

- 0: The agent is uninfected (this should be the case for any simulation)
- 1: The agent is infected but on treatment
- 2: The agent is in a primary infection stage
- 3: The agent is in a chronic infection stage
- 4: The agent is sick or in an end-stage of infection

This can be specified differently. But you must make sure that your stages are consistent across the simulation, else nonsensical things will happen. If you want a different infection stage setup, make sure your datasets and `initial_infect_stage` parameter are consistent with each other.

Dataset: `dataset_infect_stage`

This is quite a complicated dataset and is best understood by looking at the commented example in the data directory called `dataset_infect_stage.csv`. It's also included for convenience in the parameter description of `dataset_infect_stage`.

## 10.16 `_coinfect`

This is a very simple event that iterates over the living agents and sets the agent coinfect property to 1 for some agents. Users who want to model more sophisticated coinfection (e.g. TB for HIV) will likely have to write their own coinfection event.

Dataset: `dataset_coinfect`

## 10.17 `_birth`

Creates new agents with their ages set to the `age_min` property. (So if the minimum age of the simulation is, say, 15, then agents are not born as infants but as instant adolescents.)

Unless the simulation population is huge, creating agents daily (assuming the `time_step` is set to a day) makes no sense. For example consider a population of 10,000 agents with a daily time step over 20 years. On any given day, a sensible continuous random distribution of births will generate a fraction of births. But FastSTI is a discrete simulation framework and there is no such thing as a fractional agent. So instead the `birth_event_every_n` parameter must be set to how frequently the `_birth` event should be executed.

The event only creates agents every  $n_{th}$  time step. The `birth_rate` parameter is set to the birth rate and the GNU Scientific Library's `gsl_ran_poisson` function is used to determine the number of births.

Besides age, the default implementation sets the following agent properties: `sex`, `sex_preferred`, `infected`, `treated` and `resistant`.

- `sex` is set, with the proportion of males determined by the `prob_birth_male` parameter.

- `sex_preferred` is set using the `prob_birth_msw` (where `msw` stands for men who sex with women), `prob_birth_wsm` (where `wsm` stands for women who have sex with men) parameters. If the agent's sex is male, the `prob_birth_msw` value is used to determine the probability that the agent's preferred sexual partner is a female. Likewise if the agent's sex is female the `prob_birth_wsm` value is used to determine the probability that the agent's preferred sexual partner is a male.
- `infected` (i.e. the infection stage of the agent with 0 by convention meaning the agent isn't infected) is set via the dataset `dataset_birth_infect`. (See `data/dataset_birth_infect.csv` for an example.)
- `treated` (i.e. which treatment line, if any, the agent is on) is set via the dataset `dataset_birth_treated`. (See `data/dataset_birth_treated.csv` for an example.)
- `resistant` (i.e. which treatment regimens the agent is resistant to, if any) is set via the dataset `dataset_birth_resistant` (See `data/dataset_birth_resistant.csv` for an example.)

If you wish to set additional agent properties, you must provide hook in by defining a macro called `FSTI_HOOK_BIRTH_AGENT` in `fsti-userdefs.h`. The macro takes two parameters: `simulation` (a pointer to the current simulation) and `agent` (a pointer to the current agent whose property you wish to set).

Parameters: `birth_event_every_n`, `prob_birth_male`, `prob_birth_msw`, `prob_birth_wsm`,

Datasets: `dataset_birth_infect`, `dataset_birth_treated` and `dataset_birth_resistant`

## 10.18 `_report`

Reports statistics on the current state of the simulation.

This event can, and typically is, executed before, during and after a simulation. If it is executed during the simulation, the `report_frequency` parameter is used to determine how often. E.g. if set to 365, it will execute approximately once a year.

In addition to the statistics the event executes (defined in `FSTI_REPORT` in `fsti-defaults.h` which can be redefined in `fsti-userdefs.h`) you can define additional statistics using the `FSTI_HOOK_REPORT` macro.

The `results_file` parameter determines the name of the output file. By default this is left blank which means results are written to standard output.

Note that if multiple simulations are run in parallel, the output will be interleaved. At the end of the simulation you can simply sort these into the right order using the first three fields which are the simulation name, id and date of the report. You can use nearly any data manipulation tool to do this including R, Python, a spreadsheet program, or standard Posix utilities such as `sort` and `awk`.

Parameters: `report_frequency`

Datasets: None

## 10.19 `_write_results_csv_header`

Simply writes a header for the results csv file. You would only place this in the `before_events` parameter. It's clever enough to figure out that it should only write itself once to a results file.

Parameters: None

Datasets: None

## 10.20 `_write_agents_csv`

Writes the current state of every agent to an agent output csv file.

This event can be executed before, during and after a simulation. If it is executed during the simulation, the `report_frequency` parameter is used to determine how often. E.g. if set to 365, it will execute approximately once a year.

How an agent is written is determined by the `FSTI_AGENT_PRINT_CSV` macro. To write out agents differently redefine this macro in `fsti-userdefs.h`.

The `agents_output_file` parameter determines the name of the output file. By default this is left blank which means agents are written to standard output.

Note that if multiple simulations are run in parallel, the output will be interleaved. At the end of the simulation you can simply sort these into the right order using the first four fields which are the simulation name, id of the simulation, current date of the simulation and agent id. You can use nearly any data manipulation tool to do this including R, Python, a spreadsheet program, or standard Posix utilities such as `sort` and `awk`.

Also note that this file can get very large. If you have a million agents and you are running 100 simulations, it will write 100 million lines every time it is executed before, during and after each simulation. You could quickly run out of disk space. All this output also slows down simulations, so use this event sparingly.

Parameters: `report_frequency`

Datasets: None

## 10.21 `_write_agents_csv_header`

Simply writes a header for the agents csv file. You would only place this in the `before_events` parameter. It's clever enough to figure out that it should only write itself once to an agent output file.

Parameters: None

Datasets: None

## 10.22 `_write_living_agents_csv`

Exactly like `_write_agents` but only writes the living agents.

Parameters: `report_frequency`

Datasets: None

## 10.23 `_write_dead_agents_csv`

Exactly like `_write_agents` but only writes the dead agents.

Parameters: `report_frequency`

Datasets: None

## 10.24 `_write_partnerships_csv_header`

If the `record_matches`, `record_breakups` or `record_infections` parameters are set to 1, then it may be useful to write a header for the output csv file.

Parameters: `record_matches`, `record_breakups`, `record_infections`, `partnerships_file`

Datasets: None

---

## Extending FastSTI

---

FastSTI provides for a wide variety of models without any code being modified by the user. But some modellers are likely to encounter limitations. For example, they may want to model TB coinfection with HIV, or implement a cure event, or more sophisticated agent generation, or provide additional statistics.

To do this, you need to program in C. This guide assumes knowledge of C programming as well as the source code control system Git. FastSTI is developed under GNU/Linux. You need a POSIX compatible system to extend it. If you're using Windows, Cygwin should do the trick.

We recommend the following approach to developing FastSTI.

- Clone the latest version of FastSTI from the [Github repository](#).
- Use either the gcc or Clang C compiler. We also strongly recommend installing Valgrind.
- Make sure the GNU autotools are installed. Then run the following to make sure FastSTI compiles and runs properly:

```
autoreconf; automake --add-missing; ./configure; make -j check
```

- Spend time looking at the source code in the src directory to understand it better. In particular we recommend examining the following files:
  - fsti-events.c which contains the events provided by FastSTI.
  - fsti-defaults.c where all the parameters are defined.
  - fsti-defaults.h where many of the macros that can be extended or redefined are defined. Note the Hooks section. These are macros that you can define in fsti-userdefs.h (or a file included by fsti-userdefs.h) to extend existing events and other functionality.
  - fsti-defs.h where some system wide constants and structs are defined.
  - fsti-agent.h where the fsti\_agent struct is defined.
  - fsti-simulation.h where the fsti\_simulation struct is defined.

Let's say you want to write an event to coinfect agents with TB. This is what you would need to do:

- In fsti-userdefs.c write a function to do the coinfection. For example:

```

1  void tb_coinfect(struct fsti_simulation *simulation)
2  {
3      struct fsti_agent *agent;
4      // Iterate through the living agents
5      FSTI_FOR_LIVING(*simulation, agent, {
6          // Put the event logic here
7      });
8  }

```

- Put the prototype declaration for your event in fsti-userdefs.h. E.g.

```

1  void tb_coinfect(struct fsti_simulation *simulation)

```

- In the fsti-userdefs.h file you need to define the FSTI\_HOOK\_EVENTS\_REGISTER macro. This is so that your event can be invoked via the before\_events, during\_events or after\_events parameters in the input configuration file.

```

1  #define FSTI_HOOK_EVENTS_REGISTER fsti_register_add("tb_coinfect", tb_coinfect);

```

- Your event may need new parameters and datasets. It may also need new fields in the fsti\_simulation struct and new fields in the fsti\_agent struct.
  - To add new agent fields define them in the FSTI\_AGENT\_FIELDS macro in fsti-userdefs.h (or a file included by fsti-userdefs.h).
  - To add new simulation fields define them in the FSTI\_SIMULATION\_FIELDS macro in fsti-userdefs.h.
  - To provide additional parameters (or datasets) for input configuration files redefine the FSTI\_ADDITIONAL\_CONFIG\_VARS parameter in fsti-userdefs.h. An example of how to do this is already in the fsti-defaults.h file.

```

1  #define FSTI_ADDITIONAL_CONFIG_VARS(config) \
2      FSTI_CONFIG_ADD(config, example_1, \
3      "Example configuration field", 0); \
4      FSTI_CONFIG_ADD_STR(config, dataset_example_2, \
5      "Example dataset", FSTI_NO_OP)

```

Note that the name of a dataset parameter must start *dataset\_*.

- If you want to save parameters or datasets into the current simulation redefine the FSTI\_HOOK\_CONFIG\_TO\_VARS macro in fsti-userdefs.h. E.g.

```

1  #define FSTI_HOOK_CONFIG_TO_VARS(simulation) \
2      simulation->example_1 = fsti_config_at0_long( \
3      &simulation->config, \
4      "example_1")

```

## CHAPTER 12

---

### Help improve FastSTI

---

We appreciate help with FastSTI, both with coding and documentation. Here are some things that need to be done:

- **Built-in calibration:** At present if you want to calibrate a FastSTI model, you'd best use R or Python to process agent output files from FastSTI, and generate .ini configuration files as input back into FastSTI with new parameters. We think it would probably be better if FastSTI's configuration could be enhanced to calibrate the model.
- **Concurrent relationships:** Although the `fsti_agent` data structure supports concurrent relationships, none of the events are currently designed to handle meaningfully concurrent relationships.
- **More events:** Code variations on the current events and new events.
- **Build Python and R interfaces to FastSTI.**
- **Improve this documentation.**

Feel free to clone the Github repository and submit patches. But before doing so, it may be best to email [nathangeffen@gmail.com](mailto:nathangeffen@gmail.com) to inform us what you would like to do, so we can discuss the best way to proceed.



## CHAPTER 13

---

### Bugs and other issues

---

Please lodge bugs, requests for enhancements, etc at the FastSTI [Github repository issues page](#).



## CHAPTER 14

---

### References

---

FastSTI citations in academic journals:

- Geffen, N, Scholz, SM. 2018. How various design decisions on matching individuals in relationships affect the outcomes of microsimulations of sexually transmitted infection epidemics. PLoS ONE 13(8): e0202516. <https://doi.org/10.1371/journal.pone.0202516>
- Geffen, N. and Scholz, S. 2017. Efficient and Effective Pair-Matching Algorithms for Agent-Based Models. Journal of Artificial Societies and Social Simulation 20(4) 8, 2017 Doi:10.18564/jasss.348. <http://jasss.soc.surrey.ac.uk/20/4/8.html>



## CHAPTER 15

---

### License

---

FastSTI is free software licensed under the [GNU General Public License \(GPL\) version 3](#).

The license text is in the `COPYING` file of the distribution.



## CHAPTER 16

---

### Credits

---

- Nathan Geffen designed and implemented FastSTI.
- Stefan Scholz coded the syphilis model with Nathan.
- Eduard Grebe wrote the macOS installation instructions.
- Marcus Low helped get FastSTI running using Cygwin under Windows.