

# Socsim Oversimplified

Carl Mason  
carlm@demog.berkeley.edu

June 15, 2011

## Contents

<b>1</b>	<b>First Encounter</b>	<b>3</b>
1.1	The simplest possible simulation . . . . .	3
1.2	What goes on inside . . . . .	5
<b>2</b>	<b>Extending Socsim</b>	<b>6</b>
2.1	Groups . . . . .	7
<b>3</b>	<b>Supervisory and rate files</b>	<b>8</b>
3.1	Minimal .sup file . . . . .	8
3.2	Global Directives . . . . .	8
3.3	Segment specific directives . . . . .	10
<b>4</b>	<b>Population files</b>	<b>14</b>
4.1	Reckoning kinship . . . . .	14
4.2	Reference to marriages . . . . .	15
4.3	Reference to transition history . . . . .	16
<b>5</b>	<b>The marriage file</b>	<b>17</b>
5.1	Reckoning marriages . . . . .	17
<b>6</b>	<b>Transition history files</b>	<b>20</b>
<b>7</b>	<b>Specifying demographic rates</b>	<b>21</b>
7.1	Structure of vital rates . . . . .	22
7.2	Mortality rates . . . . .	22
7.3	Marriage rates . . . . .	24
7.4	Divorce rates . . . . .	24

7.5	Fertility rates . . . . .	24
7.6	Transition rates . . . . .	25
7.6.1	Duration specific transition rates . . . . .	25
7.7	Event competition . . . . .	27
7.8	Generating potential waiting times . . . . .	27
<b>8</b>	<b>Heterogeneity multipliers</b>	<b>28</b>
8.1	Fertility multiplier (fmult) . . . . .	28
<b>9</b>	<b>The Marriage Market</b>	<b>29</b>
9.1	Marriage mechanics . . . . .	29

## 1 First Encounter

Although Socsim has many charms, ease of use is not one of them. Originally written in the 1970s, Socsim continues to develop as a research tool constantly changing to meet the goals of new research projects. A Slick graphic user interface has never been part of the plan<sup>1</sup>.

Raising the barrier to first use still higher is Socsim's insatiable appetite for input data. Because micro-simulation is about simulating a whole mess of individuals' vital demographic events, Socsim needs a whole mess of age, sex, marital status, *group*<sup>2</sup> and parity specific rates for births, deaths, marriages and change of group membership. There is of course, a default structure – for example if mortality rates are not specified for married males, Socsim will by default use those of single males of the same group. Or if parity specific fertility rates are not specified, Socsim will use the same rates for all parities. Still a moderately complicated simulation will still require an *astounding* number of rates.

### 1.1 The simplest possible simulation

To run Socsim you must have the following stuff at a minimum:

**A compiled version of Socsim.** See `lab.demog.berkeley.edu/socsim` if this the IT fairies have not already handled this for you.

**An initial population and marriage file.** Socsim reads and writes Population data in a specific format. Population (or *.opop* files) are further described in Section 4. In order for Socsim to start, it must find an *initial population* stored in this format. Generally individuals in this population will not be married, but an empty marriage file (`something.omar`) is still required. The initial population need have no particular age or sex distribution<sup>3</sup>, since one usually runs Socsim for a couple of hundred (simulated) years in order to convert the initial population into a population with a known and stable age structure before simulating the processes that are of real interest. This can be done differently, but not without added complication.

---

<sup>1</sup>to the extent that there is a “plan”

<sup>2</sup>Groups are a additional category that can be used to extend Socsim in interesting ways. Groups are described in a little more detail in Section 2.1

<sup>3</sup> One can start Socsim with an initial population complete with a kin network, but typically the purpose of this whole exercise is to generate a simulated a kinship network – if you already have the kinship data then maybe you should stop wasting time and write your dissertation

**A *supervisory or .sup file.*** The supervisory file generally named `something.sup` is passed to Socsim on the command line. It must contain certain ***global parameters*** such as the number of simulation ***segments***, name patterns of initial population and final population files, and either a complete set of vital demographic rates or else ***include*** statements specifying where to find files containing those rates. The full explanation of ***.sup*** file is in Section 3.

The minimal file, (from the `Sample` directory of the source code tree) is shown in Figure 1

**vital demographic rates** As noted previously, the vital demographic rates can be included within the supervisory file. But generally it is more convenient to store them in one or more separate files which are referred to in the supervisory file with an ***include*** directive.

The format in which the rates are given to Socsim is critical. Generally a set of rates for particular demographic event begins with an identifier such as:

**birth 1 F married 0**

which indicates that the following several lines will contain a complete set of age specific fertility rates for group 1 married females of parity 0.

Following the rate identifier will be one or more lines each of which contain three space separated numbers. The first two indicate year and months of the **upper age bound** over which the rate is in effect. The two quantities are added together so “1 1” is equivalent in effect to “0 13”. Note that the first age category specified should NOT have “0 0” as its upper bound. Age categories are otherwise flexible, one can specify some age categories that are as short as a single month, while others are many years or decades wide.

The rate, given in per month terms, is the third number. Socsim insists that there be a line specifying an age category with an upper bound of 100 years. Obviously this last rate can be 1 or zero. For more details on specifying rates See Section 7

When all of the above bits are in place, Socsim can be executed from the Unix shell:

```
/path/to/socism supervisoryfile.sup 12345
```

The result will be a considerable amount of screen output indicating which options are set and how the simulation is proceeding. Or just a short complaint about an inconsistency in the various input files.

When complete, a logfile called `/usersupervisoryfile.sup12345.log` should be evident in the current directory as well as an output population `.opop`; marriage, `.omar` and perhaps a extra variables, `.opox` file whose locations depend on the `output_file` directive in the `.sup` file.

The `.opop`, `.omar` and `.opox` files produced by a Socsim run, are of course suitable initial population files(s) for subsequent Socsim simulations. Eventually however, you will probably have to stop simulating and start analyzing those files. See Sections 4 and 5 for details on how these files are structured.

## 1.2 What goes on inside

While all that “information” is scrolling past on the screen and filling the log file, Socsim is busily scheduling and executing vital demographic events for each simulated being in the initial population as well as all of their descendants.

Socsim begins each simulation *segment*<sup>4</sup> by scheduling an event for each “living” person. At all times during the simulation every non dead simulated being has exactly one scheduled event. Whenever an event is executed or a person’s marital status or parity changes, a new event is scheduled for that individual. Thus death of a spouse causes a new event to be scheduled for the widow; a birth causes a new event to be scheduled for both the child and the mother.

To determine which event a person is to be scheduled for, Socsim generates a random waiting time for each event that the person is at risk of having. Obviously the user supplied age, sex, group, and marital status specific rates govern the process of random waiting time generation. Once all of the potential events have randomly generated waiting times associated with them, the earliest one is chosen and that event is scheduled. Information on the other possible events is discarded. In short, it’s a competing risks model. The probability of each event is independent of all other events and the earliest event is the only one that counts. See Section 7.7 for more details.

---

<sup>4</sup>A simulation segment is a period of simulated time during which a single set of vital rates are in effect

Once each person has a scheduled event, Socsim starts to march through time. A list is created of all of the events scheduled for the first month of the simulation segment and each is drawn in random order and executed. When the execution of an event causes a new event to be scheduled (as most do) the resulting new event is either placed on the calendar for execution in a future month or if the waiting time is zero, it is inserted in the list of events waiting for execution in the current month and is drawn in random order.

When all of the events scheduled for the current month are executed, Socsim increments the month and repeats the event execution procedure just out lined. When the last month of the segment is completed, Socsim begins the next segment by reading the rate files and generating new events for all living people. If there is no subsequent segment, then Socsim finishes by writing the population (.opop), marriage (.oar), extra variables (.opox), and population pyramid (.pyr) files.

## 2 Extending Socsim

Socsim is open source so modifying the code to take account of cultural norms, ethnological truths, or behavioral theories is encouraged. Since many Socsim based research projects require some kind of enhancement, the program is structured so as to encourage modification. It is possible to enhance Socsim without first gaining a deep knowledge of all the intricacies of the program by taking advantage of “hooks” that have been placed in the code and which call functions at key points in the execution. “Stub” versions of these functions can be found in the `enhancementNULL.c` file in the top of the source tree.

When Socsim is compiled with the proper command line arguments, functions in `enhancement.c` become part of the code. If your modifications can be contained within `enhancement.c`, then improvements and bug fixes to the main trunk of the code will not conflict with your project – probably.

Before you consider enhancing Socsim, however, you should first give a lot of serious thought to how you might manage to accomplish your goal **without writing any C code**. One approach for doing this is to try to make clever use of the concept of *groups*, described below.

Another approach to email Carl Mason ([carlm@demog.berkeley.edu](mailto:carlm@demog.berkeley.edu)) and see if maybe he’ll do the programming for you.

## 2.1 Groups

Socsim implements *groups* by simply adding a variable to the population list. Every person is a member of exactly one group at all times. Group membership is determined at birth according to the rule given in *child\_inherits\_group* directive, but can change at any time according to age, sex, marital status, and group specific *transition* rates that the user specifies.

The group designation can be thought of as ethnicity, location, health status, wealth, education or any characteristic of a person that might or might not change according to a rate schedule.

Members of each group are subject to the same rates for vital events. That is the members of group 7 have fertility, mortality and nuptiality schedules that are independent of those of group 8. When an individual transitions from one group to another, she becomes subject to the new group's rates and therefore must have a new event scheduled.

Socsim determines how many groups are in a simulation at the beginning of each segment by looking at both the starting population and the rates. If Socsim finds a person of group N in the initial population than it knows that there are at least N groups in the simulation. If it finds a mortality rate for members of group Q then it knows that there are at least Q groups.

Because of the default pattern of rates described in Section 7, it is possible to add groups to a simulation and only specify the rates for the group which differ from the those of the existing group.

### 3 Supervisory and rate files

The name of the *supervisory file*, generally with `.sup` suffix, is passed to Socsim on the command line. Socsim expects such a filename followed by a random number seed and will not run without these two command line arguments.

What Socsim expects to find in the supervisory file is a set of parameters and possibly rate specifications that allow it to run the simulation. The supervisory file has to either contain all of the information Socsim needs (aside from the random number seed) or else it must have *include* directives that tell Socsim where else to look. The `.sup` file must consist entirely of valid directives or comment lines. Comments are lines that begin with `'*`. Comments are ignored by Socsim.

The `.sup` file includes both global (affecting the entire simulation) and segment specific simulation parameters. The `run` directive indicates the end of a set of segment specific parameters. When Socsim encounters a `run` directive, it stops reading the `.sup` file and executes the simulation segment. When the segment's execution is complete, Socsim returns to reading the `.sup` file where it left off. This has two important implications:

1. If the directives for a segment are not followed by a `run` directive, the segment will not be executed. Socsim will simply read the instructions for the next segment and execute those – assuming that they end with a `run` directive.
2. Errors in a `.sup` file will not be caught until they are encountered. If there is an error in the specification of the 92<sup>nd</sup> segment, Socsim will execute the first 91 segments before it exiting abnormally.

#### 3.1 Minimal `.sup` file

At a minimum the supervisory file must include only a few directives. Figure 1 shows a minimal but sufficient `.sup` file.

#### 3.2 Global Directives

Global directives affect the entire simulation and do not change with each new segment.

1. `segments n` : `n` is the number of simulation *segments* in the simulation run. A segment is a period of time under which a given set of vital rates are effective. Every rate change implies a new segment.

---

```

*****
** This is the simplest possible socsim .sup file. It will run a one
** segment simulation with starting population in test.opop and
** test.omar and ending population in test.out{.opop,.omar} The
** duration of the one and only "segment" is 1200 months; the minimum
** birth interval is 24 months; heterogenous fertility is turned off;
** rate files are in ratefile.Lese.
**
** type [path to socsim] run.sup 12345
** to run socsim
*****
segments 1
input_file test
output_file test.out
duration 1200

include ratefile.Lese
run

```

Figure 1: Minimal supervisory file

---

2. `input_file input-file-stem` : indicates where Socsim should look for the input files. `input-file-stem.opop` will be the initial population file; `input-file-stem.omar` will be the initial marriage file. The initial marriage file is necessary even if the initial population has no kinship structure. In this case, however, it should be an empty file.
3. `output_file output-file-stem` : indicates where Socsim should write output files. Just as with `input_file` directive, Socsim expects the argument to be a file stem, from which Socsim can construct a complete path by simply appending a suffix. By default, the population and marriages files are only written at the end of the final segment and are called `output-file-stem.opop` and `output-file-stem.omar`. However, The `write_output` directive below causes these files to be written at the end of a segment and modifies naming conventions to

avoid overwriting.

### 3.3 Segment specific directives

1. `duration n`: where `n` is the number of months that the current simulation must run.
2. `bint n`: `n` is the minimum birth interval. As a concession to reality, Socsim can impose a minimum number of months between births. For humans 10 months is a good number to use. Socsim adjusts the specified fertility rates upward to compensate for the birth interval.
3. `hetfert 1/0`: 1 to enable the heterogeneous fertility feature 0 to disable it. If enabled, each female will have a gamma distributed random variable by which her fertility is multiplied before random waiting times are generated. The result is a wider variation in sibling set size than would otherwise result. Note that the degree to which this fertility “multiplier” is inherited by daughters can be set by the user see *alpha* and *beta*.
4. `alpha x` and `beta y` where `x` and `y` are real numbers that determine the degree to which the fertility multiplier is inherited from each woman’s mother (*alpha*) and the shape of the beta distribution from which the fertility multipliers are drawn. See Section /reffertmult for a complete description.
5. `endogamy -1..1`: determines how suitors from other *groups* are treated in determining suitability for marriage. When ego is having a marriage event executed, s/he inspects everyone on the marriage queue of the opposite sex. See Section 9 for details on the workings of the marriage market.

A value of `endogamy` between 0 and 1 is taken as the probability that a potential spouse who is a member of a different group from that of ego will be rejected. A value of 1 therefore implies *endogamy* while a value of 0 implies that group membership will not matter with respect to marriage.

A value of `endogamy` between -1 and 0 is taken as the negative of the probability that a potential spouse from the **same group** will be rejected. A value of -1 therefore enforces complete *exogamy* – all suitors of ego’s own group will be rejected with probability 1.

6. `marriage_agedif_min` `n` and `marriage_agedif_max` `n` determine bounds on the permissible age difference between spouses.

$$\text{marriage\_agedif\_min} \leq a \leq \text{marriage\_agedif\_max} \quad (1)$$

$$\text{where } a = \text{groom's age} - \text{bride's age} \quad (2)$$

potential spouses who are not within the specified age range are excluded from marriage. the default values -120 (bride cannot be more than 10 years older than groom) and 120 (groom cannot be more than 10 years older than bride).

7. `marriage_peak_age` `n` determines (together with *marriage\_slope\_ratio*) the preference for age difference (in months) among spouses. After incest and endogamy/exogamy checks are complete, suitors who have not been eliminated, are assigned a score based on the groom's age - bride's age. If this age difference is greater than `marriage_peak_age` than the score declines at a constant rate as the difference increases. If the age difference is less than `marriage_peak_age` then the score declines at `marriage_slope_ratio`\* that constant rate as the age difference decreases.

Before giving yourself a bald spot over this, note that only the ordinal rank of this score matters because the spouse is chosen randomly only from among the maximum scoring suitors. The idea is simply to favor a certain optimal age difference and to allow for a preference for older or younger spouses. In most simulations the marriage rates – which determine the age at which people initiate a marriage search, are **far** more important in determining marriage patterns than are these marriage preference parameters. The default value for `marriage_peak_age` is 36 months. See Section 9 for details on the workings of the marriage market.

8. `marriage_slope_ratio` `w` orks with `marriage_peak_age` to determine a marriage preference “score” for potential spouses. A value of 1.0 implies an equal preference score for an age difference that is x months from `marriage_peak_age` regardless of whether the groom or bride is older. A value greater than 1 implies a that the preference score will decline faster as the brides age increases relative the the grooms – than it will as the groom's age increases relative to the bride's. The default value, 2, means that preference scores decline

twice as fast with distance from `marriage_peak_age` as the bride's age increases relative the the groom's.

9. `child_inherits_group rule` determines how group membership is assigned at birth. Socsim understands the following rules:
  - `from_mother`
  - `from_father`
  - `from_same_sex_parent`
  - `from_opposite_sex_parent`
  - `n` where `n` is a group to which all new borns are assigned.

By default, Socsim assigns newborns to mother's group. That is, if no option is specified, Socsim will use "from\_mother".

10. `duration_specific transit group sex marital_status target_group` : tells socsim to treat the specified rate block as duration rather than age specific. Note that this directive does not take the place of the keywords that introduce the rate block. To define a duration specific group transition, something like the following is required:

```
duration_specific transit 4 M single 18
transit 4 M single 18
5 0 0.0
5 10 0.5
0 1200 0.0
```

For details see Section 7.6.1.

11. `hettrans 1/0` : enables heterogeneous transition probability. It only makes sense to set this to 1 if you are working with an enhanced version of Socsim that defines how this should work.
12. `parameter0..parameter5 r` : `parameter0` through `parameter5` should be set only if the enhanced version of Socsim that you are using defines what they do.
13. `read_xtra 1/0` 1 to make Socsim read a file called `input-file-stem.opox` or crash if the file does not exist. The `.opox` file should contains a set of `extra variable` values for each person in the initial population. This will generally only be used in versions of Socsim that have been

modified. See Section ?? for description of the programming hooks available for “easy” modification. The plain vanilla version of Socsim does not use extra variables so reading them will not generally cause anything useful to happen.

14. `size_of_extra n`: `n` is the number of variables that should be read for each person from the .opox file. This will generally be made to default when one modified Socsim. In its unmodified state, Socsim does not use extra variables.
15. `include filename`: Socsim will read and parse `filename` as if it were part of the current file. It is strongly recommended that `include` be used to keep rate specifications separate from the rest of the simulation parameters. `includes` can be nested. See Section 7 has details on how vital demographic rates are specified.
16. `execute string`: `string` is passed to the Unix shell and its output is routed to the screen. After `string` exits without error, Socsim returns to processing the current file.

It is possible with this directive to call an external program that might perhaps generate rate sets on the fly perhaps in response to the previous simulation segment. For example:

```
execute generate_rates 1 5 0 >mortality.seg4
include mortality.seg4
```

## 4 Population files

Socsim reads and writes everything it knows about people in two files: the population *.opop* file and the marriage *.omar* file<sup>5</sup>. Both are space delimited files and both contain only numbers.

Since one generally runs Socsim for 200 simulated years in order to start from a population with a known and stable age structure, it is seldom necessary to construct an initial population with any information other than the age and sex of a small number of individuals. Such a file can be easily constructed in a spreadsheet program. The corresponding marriage file is simply an empty file with correct name.

One needs to come to terms with the structure of the *.opop* and *.omar* files in much more detail when analyzing simulation output. A snippet of code for reading an *.opop* file into R is given in Figure 2. In terms of R the *.opop* file's contents fit naturally into a *data.frame* with 14 columns all of which are numerical. In more general terms, the *.opop* file is a matrix where each row contains information on a single person and each column contains a particular bit of information on each person.

Although its structure suggests that a *.opop* file might be right at home in a spreadsheet program, this is not so. First the files tend to be too large since they include not only a row for each person who ever lived. Even a modest sized simulation can easily have 100,000 rows. But more important, much of the information in the *.opop* file consists of identification numbers of other people. In other words the *opop* file is a multiply linked list. For manipulating linked lists, spreadsheets are profoundly suboptimal.

In most cases, the *.opop* file will be sorted in order of person id and since person ids are sequential integers assigned in birth order this means that the *.opop* file is generally sorted in birth order, with the row number often being the same as the person id. This is not however guaranteed to be the case so check that it is so before relying on it<sup>6</sup>.

Table 1 shows which information is in each column.

### 4.1 Reckoning kinship

In analyzing Socsim output, one is often interested in reckoning kinship. Since with the possible exception of the initial population, everyone in socsim is related to everyone else, it is possible to find nearly any two people's

---

<sup>5</sup>It may also read and write a file of extra variables and a file of transition history

<sup>6</sup>It is best practice not to rely on the *.opop* file's pid ordering because this convention can be broken in the initial population file

---

```
## read .opop into dataframe the .opop file contains one row for each
## simulated person who ever "lived". It generally includes many who
## "died" long ago.

opop<-read.table(file="../SimResults/example.opop",header=F,as.is=T)

## assign names to columns
names(opop)<-c("pid","fem","group",
              "nev","dob","mom","pop","nesibm","nesibp",
              "lborn","marid","mstat","dod","fmult")
```

Figure 2: Rcode for reading simulation output files

---

relationship by following the chain of parents and siblings. To find ego's maternal grandmother, one simply finds the ego's mother's person id in the 6<sup>th</sup> column of ego's row in the opop file. Moving then to the row of the opop file corresponding to ego's mother's person id, one look's again in the 6<sup>th</sup> column to find egos' mother's mother's person id.

To find all of ego's children, one starts with the person id of ego's last born child (stored in column 10) of egos' row of opop. In the row corresponding to ego's last born child's row of opop, we find, in column 8 (9), ego's last born child's next eldest sibling through her mother (father). In that person's row of the .opop file, we can find yet another next eldest sibling and so on until we find ego's first born child, whose next eldest sibling through mother (assuming that ego is female) is necessarily zero.

Alternatively, one could simply collect all the rows of opop which have that same value in column 6 (mom) and or 7(dad) as ego has.

The R computing environment (<http://www.r-project.org>) is particularly well suited for doing this kind of analysis of kinship.

## 4.2 Reference to marriages

Since individuals can be married more than once (simultaneously in some cases) reckoning marriage information is trickier than working with kinship alone. See Section 5 for more details on how to work with Socsim's .omar

position	name	description
1	pid	Person id unique identifier assigned as integer in birth order
2	fem	1 if female 0 if male
3	group	Group identifier 1..60 current group membership of individual
4	nev	Next scheduled event
5	dob	Date of birth integer month number
6	mom	Person id of mother
7	pop	Person id of father
8	nesibm	Person id of next eldest sibling through mother
9	nesibp	Person id of next eldest sibling through father
10	lborn	Person id of last born child
11	marid	Id of marriage in .omar file
12	mstat	Marital status at end of simulation integer 1=single;2=divorced; 3=widowed; 4=married
13	dod	Date of death or 0 if alive at end of simulation
14	fmult	Fertility multiplier

Table 1: contents and format of the .opop file

file. For the present purpose note that column 11 of the .opop file contains a pointer, in the form of a marriage id number, to ego's most recent marriage. If column 11 is zero, then ego has never been married.

### 4.3 Reference to transition history

If the simulation includes group transitions, than socsim will write a file with the same path as the population and marriage files but with the suffix *.otx*. The transition events do not have unique ids as do marriages, but each transition record contains the person id of the protagonist. Consequently it is much more natural to link from the transition history to the population file.

Section 6 describes the *otx* file in detail.

## 5 The marriage file

The marriage (.omar) file is similar to the population (.opop) file in that much of the information it maintains is in the form of unique id numbers that correspond to rows of the .opop file or to the .omar file itself. In other words, the .omar file is another *linked list*

Figure 3 shows a snippet of R code suitable for reading a .omar file. Table 2 shows the meaning of each column of the file.

### 5.1 Reckoning marriages

As with the .opop file, reckoning marriage histories requires following a list of integers from one record (row) to another. The marriage id of the wife’s *most recent prior* marriage is stored in column 7. The corresponding pointer for the husband is stored in column 8. The other six columns of the marriage record hold information on the marriage itself. Note that marriages are created sequentially, but monogamy is not assumed. So for any particular marriage record, the wife’s prior marriage (pointed to in column 7) must have a start date (column 4) that is no larger (later) than that of the particular marriage. A “prior marriage” must start before (or in socsim, at least in the same month) as the subsequent marriage. But the same is not true for the marriage end date, which is stored in column 5.

Also, be careful of zeros. Marriages that remained intact when the simulation ended have end dates of zero.

To find a female ego’s first husband, one begins with the .opop file. The 11<sup>th</sup> column of ego’s row holds the marriage id of egos most recent marriage. The row in the .omar file whose first column entry matches that number is the record of egos’ most recent or “last” marriage. In order to find her **first** marriage, we must locate the row of the .omar file wherein the entry in column 1 matches the the marriage id of the wifes prior marriage which is stored in column 7 of the .omar file. We repeat this proces until we locate a record of a marriage for which ego’s person id is stored in the second column and column 7 holds a zero.

In R, a more efficient way of finding first marriages is to select the subset of marriages for which either the husband’s prior or wife’s prior marriage (column 7 or column 8) are zero, and then use the `match()` function to link the marriage id to the husband or wife’s opop record. Figure 4 shows a snippet of R code that performs this task.

---

```

omar<-read.table(file="../SimResults/example.omar",header=F,as.is=T)
names(omar)<-c("mid","wpid","hpid","dstart","dend",
              "rend","wprior","hprior")

rownames(omar)<-omar$mid

```

Figure 3: R code for reading a .omar file

position	name	description
1	mid	Marriage id number (unique sequential integer)
2	wpid	Wife's person id
3	hpid	Husband's person id
4	dstart	Date marriage began
5	dend	Date marriage ended or zero if still in force at end of simulation
6	rend	Reason marriage ended 2 = divorce; 3 = death of one partner
7	wprior	Marriage id of wife's next most recent prior marriage
8	hprior	Marriage id of husband's next most recent prior marriage

Table 2: Structure of Socsim marriage file

---

```
## get first marriage id -- socsim stores marriage ids as linked list
## headed by most recent marriage. The (h/w)prior field stores the id
## of each spouses prior marriage

## select a subset of marriages which are first for at least one partner
fomar<-omar[omar$hprior == 0 | omar$wprior == 0,]
opop$fmid<-NA

## use match() to lookup the marriage id of each person (in opop)'s
## first marriage id
opop[opop$fem==0,"fmid"]<-
  fomar[match(opop[opop$fem==0,"pid"],fomar$hpid),"mid"]
opop[opop$fem==1,"fmid"]<-
  fomar[match(opop[opop$fem==1,"pid"],fomar$wpid),"mid"]
```

Figure 4: R code for finding first marriages

## 6 Transition history files

If group transitions are part of the simulation then socsim will write (and may read) a transition history file. In spirit, the *otx* file is much like the marriage file, the main differences are that the *file* is linked to only one person and there is no unique identifier for a transition history record.

Like all other Socsim output files, the *otx* file is space delimited and contains only numbers. Table 3 shows the structure of the *otx* file. Figure 5 contains R code for reading an otx file into a data frame.

position	name	description
1	pid	Person id to who the transition event occurred
2	date	Month in which the transtion occurred
3	fromg	group from which the person transitions
4	tog	group into which the person transitions
5	sequence	a non positive number indicating the order of the event. A zero indicates that the current record refers to the most recent transition event; a -7 indicates that seven transitions have occurred to this person subsequent to that of the current record.

Table 3: Contents and format of the .otx file

---

```
otx<-read.table(file="../SimResults/test.otx",header=F,as.is=T)
names(otx)<-c("pid","month","fromg","tog","pnum")
```

Figure 5: R code for reading an *otx* file

## 7 Specifying demographic rates

It is most convenient to store demographic rates for each simulation segment in distinct files and use the `include` directive to reference them from the supervisory file (See Section 3).

To run a moderately realistic simulation, Socsim requires age specific fertility rates for females and mortality rates for both males and females of each *group* and marital status. If rates do not differ by marital status or group then you can use Socsim’s default rules to avoid entering rates repeatedly. Figure 7 shows the rules that Socsim uses when it encounters incomplete rate sets. The “`==;`” symbol in Figure 7 means “defaults to” so for example,

```
widowed ==> divorced; parity 0; group 1
```

which appears in the “Fertility Rates” section under the heading “For parity zero women in group 1” indicates that if Socsim does not find fertility rates for widowed females in group 1 it will Where rates default to “Zero” it use the rates for divorced women of the same group and parity.

Where Figure 7 indicates that a rate block defaults to “Zero”, Socsim does not default to anything leaving such events with zero probability of occurring at any age. So for example, unless you think that single males in group 1 should live forever, you must specify mortality rates for such “people”.

## 7.1 Structure of vital rates

The format in which Socsim expects to find rates is simple. Each

**block** of rates begins with a set of keywords which indicate the event for which the rates apply and the marital status, sex, group membership and possibly parity of the people who are at risk of experiencing the event. That information is followed on subsequent lines by rate values and the **upper bound** of the age group for which the rate is in force.

A **rate block** is a complete set of age specific rates governing a demographic event for people of a particular sex, group and marital status. An example of a rate block is shown in Figure 7.1. The first line after the comment line, indicates which event (death); group (1); sex (M=male); and marital status (single) this rate block pertains to. The order matters and is always, Event then group then sex then marital status. In the case of birth rates this may be followed by number indicating parity. In the case of transition rates, the line must end with a number indicating the destination group.

Each subsequent line contains a one month rate (in the case of fertility) or a one month probability in the case of all other events and the age interval over which the monthly rate (probability) holds. The first two numbers in the line are years and months of the upper age bound. These are added together so a 1 and 11 would mean 23 months. The third number is the rate. In the case of fertility it represents the expected number of births **per month** to a woman who survives to at least the beginning of the interval that ends at the given upper age bound and begins one month after the upper age bound given on the previous line.

For all other events it is the probability that the event occurs to a person during any given month in that age interval. Thus in Figure 7.1 the probability that a single male dies between the ages of 1141 and 1200 months, conditional on having survived to the beginning of the age interval is  $1 - (1 - .08326)^{60}$ .

## 7.2 Mortality rates

Mortality rates are the most straight forward of the rates that Socsim uses. The example in Figure 7.1 is typical. The event identifier is **death**, the “1” refers to the group, “M” to male and “single” is of course marital status.

Figure 6: Rate default rules

---

**Fertility Rates**

For parity zero women in group 1

-----  
 single ==> Zero  
 married ==> Zero  
 divorced ==> single; parity 0; group 1  
 widowed ==> divorced; parity 0; group 1  
 cohabiting ==> married; parity 0; group 1

For women in group 1 with higher parity

-----  
 mstatus m; PARITY P; group 1 ==> mstatus m; PARITY P-1; group 1

For women of any parity and any group > 1

mstatus m; parity p; GROUP G ==> mstatus m; parity p; GROUP G-1

**Marriage, Divorce and Mortality Rates**

For men and women in group

-----  
 DEATH for single; sex s; group 1 ==> Zero  
 MARRIAGE for single; sex s; group 1 ==> Zero  
 DIVORCE for mstatus m; sex s; group 1 ==> Zero

(Events except for divorce)

-----  
 event e for divorced;sex s ==> event e; for single;sex s;group 1  
 event e for widowed;sex s ==> event e; for divorced;sex s;group 1  
 event e for married;sex s ==> event e; for widowed;sex s;group 1  
 event e for cohabiting;sex s ==> event e; for married;sex s;group 1

For Groups > 1

-----  
 event e for mstatus m;sex s; group g ==> event e for mstatus m;sex s; group g-1

**Group Transition Rates**

TRANSITION to group H for mstatus m;sex s; group g == Zero

---

Figure 7: Example of a block of mortality rates

```

*Mortality, single Male (lines beginning with * are comments)
death 1 M single
0      1      .0460940
0      12     .0057540
0      60     .0008730
0      120    .0002600
.
.
.
0      1140   .0832630
0      1200   .0832630

```

### 7.3 Marriage rates

Marriage rates are specified with the event identifier of `marriage`, but it should be born in mind the event which these rates regulate is not really marriage but rather the commencement of a marriage search. Since marriage requires two participants Socsim cannot simply execute a marriage when the event is scheduled. Marriage can only happen when two potential spouses with pending marriage searches find each other in the *marriage queue* See Section 3 for other important features of the marriage process.

As a consequence, marriage rates often need to be raised in order to achieve the specified result.

### 7.4 Divorce rates

Divorce rates are specified with the event identifier `divorce`. Divorce is unusual among Socsim events in that it's rates do not apply to the age of one spouse or the other but rather by age of the marriage. It is thus generally not necessary to specify divorce rates for both sexes.

### 7.5 Fertility rates

Fertility rates are specified with the identifier `birth` and are different from other rates in two ways:

1. They are parity specific. But they default to parity  $n - 1$  so it is only necessary to specify rates for parity zero.
2. They are rates rather than probabilities. So multiplying a rate by the number of months in the age category gives the expected number of births that a woman who lives through the age category will experience.

## 7.6 Transition rates

Transition rates give rates of “transition” from one *group* to another. By default, no transitions occur, however, if the initial population contains more than one group then the group inheritance rule determines the group identity of new borns.

Transition rates are specified with the identifier `transit` and are different from other rates in that both the group to which the rate applies **and** the group to which the event will cause a person to belong must be specified.

To specify transition rates from group 1 to group 2 for single males, one would write the following:

```
transit 1 M single 2
```

As noted in Figure 7, transition rates have no defaults. All rates have to be specified in order to take effect.

### 7.6.1 Duration specific transition rates

It is often useful for transition rates to be duration rather than age specific. In other words, the probability of a transition event occurring can depend on the time since the individual transitioned into the current group rather than the individual’s age. In order for this distinction to matter, a person must have experienced at least one transition other wise the time spent in the group is equivalent to the person’s age.

Divorce rates are always duration specific, but transition rates may vary with either age or duration. The format for specifying transition rate blocks is the same regardless of whether they are age or duration specific. To tell Socsim that a particular rate block is meant to be duration specific, add the directive: `duration_specific` as in:

```
duration_specific transit 4 F married 5
```

when this directive is encountered, the log file will indicate that transition rates from group 4 to group 5 for married females will be duration rather than age specific. The default is for transition rates to be age specific so no indication is given in the log file of that condition. It is possible to have both age and duration specific transition rates in effect in the same simulation, however, only one transition rate block is allowed per pair of groups. So transitions from say group 3 to group 5 for single males must be *either* age or duration specific, but whichever it is, has no effect on transitions from group 3 to group 5 for *married* males. One may be age specific while the other may be duration specific.

As noted in Section 3.3, the `duration_specific` directive does not replace the keywords that define the rate block.

## 7.7 Event competition

At any moment during the simulation every living individual has exactly one future event scheduled<sup>7</sup>. Future events are stored in a 1200 element array of linked lists<sup>8</sup>, each element of the list represents a “month” of Socsim time.

Socsim executes the events scheduled for the current month, then increments the clock and moves to the next month. Events always affect the individuals for whom they are scheduled but may also affect spouses and others to whom they are connected. All births, deaths, marriages and group transitions that are scheduled for the current month are executed in random order. After a person’s event is executed, unless that event was death, a new event for that individual must be scheduled. If the event triggers a status change, for example if ego is married, then her death causes her spouse to become a widow. Since widows may have different event probabilities, a new event must also be generated for those who become widows.

New events are scheduled by an “event competition.” This event competition is also held once for each living person at the beginning of each simulation segment (that is, every time the demographic rates or societal constants change).

Each event for which the individual is at risk (e.g., men rarely give birth) can be modeled as a piecewise exponential distribution. A random number is used to generate a waiting time until this event occurs (which is bounded by the individual’s maximum possible age at death). The individual’s next event is the one with the shortest randomly generated waiting time. The event competition thus follows a competing risk framework wherein the probability of each event is independent of all others.

## 7.8 Generating potential waiting times

The waiting time algorithm is conceptually equivalent to drawing a random number  $u$ , from a uniform (0,1) distribution, calling  $u$  the probability that the event will not yet have occurred, then finding the first month by which the probability of non-occurrence is less than or equal to  $u$ . The probability that an event will not have occurred by a particular month  $T$  is given by the expression

$$\prod_{t=0..T} (1 - p_t) \tag{3}$$

---

<sup>7</sup>Technically, at a given moment, one person *could* be having an event executed and could thus not yet have her next event scheduled.

<sup>8</sup>Usually Socsim recognizes 1200 future months but this can be altered

Where  $p_t$  is the probability of the event's occurrence in period  $t$  conditioned on it not having occurred at any time before  $t$ . Since  $(1 - p_t)$  is always between 0 and 1, the expression given above is nonincreasing in  $T$ . Consequently, beginning with  $t = 0$  we can successively multiply the  $(1 - p_t)$  terms together until the value of the product falls below  $u$ . What Socsim does is mathematically equivalent to this procedure, however, the implementation in function `datev` takes advantage of fact that the probabilities can be the same over months or years and works with powers of  $(1 - p_t)$ .

## 8 Heterogeneity multipliers

Heterogeneity beyond that which follows from the algorithm described in Section 7.8 is often desirable in microsimulation. Socsim increases heterogeneity of fertility for example, in order to create more realistic sibling set sizes and to allow for heritability of fertility.

Heterogeneity of mortality and group transitions are not included by default but much code is in place to allow users to add these features easily in a way that makes sense for a particular simulation.

The general principle is that if each person at risk of an event is given a value  $\theta$  drawn from a distribution with mean=1 and the hazard rates used for generating each individual's waiting time for the event are multiplied by  $\theta$ , then  $E(h\theta) = h$ . Where  $h$  is the original hazard of the event. Thus the overall population's event history should still reflect the rate structure but with greater variance/heterogeneity.

### 8.1 Fertility multiplier (`fmult`)

When the heterogeneous fertility option, (`hetfert`) is enabled, each female in the population is assigned a fertility multiplier. For the initial population these multipliers may be read in the `.opop` file or they may be generated by Socsim. Females born during the simulation will have multipliers generated by Socsim. Once assigned, fertility multipliers remain with the woman for her entire life increasing or reducing proportionally the hazard of giving birth at each.

In the current implementation, individual fertility multipliers, `fmults`, are pseudorandomly distributed as a cubic approximation to the beta distribution with mean 1.0, variance 0.416, and a range of 0 to 2.4.

## 9 The Marriage Market

Because two prospective spouses may not arrive at the altar in the same month, SOCSIM can only schedule the beginning of a "marriage search" rather than a marriage per se. This is an inescapable consequence of the need for two spouses and of SOCSIM's being a closed simulation. When what we call a "marriage" event comes up for execution, ego initiates a marriage search which results either in (1) the selection of the "best" match from among those who are both eligible and available, or (2) ego joining the *marriage queue* of available prospective spouses. In this latter case, ego's marriage will only be completed when ego turns out to be eligible and best for some other individual of the opposite sex whose marriage event is executed in the future.<sup>9</sup>

Thus rather than finding, by some criteria, the globally optimal groom for each prospective bride (or vice versa). SOCSIM's marriage algorithm simply allows each ego to make her best choice from those who are also seeking marriage at the same time. If by spending time on the marriage queue, individuals become more suitable than they originally were, that is too bad. If individual preferences change as a result of the sex balance in the population, that too is unaccounted for in the default version of SOCSIM<sup>10</sup>. Regrettably, when populations are small or sexually unbalanced, SOCSIM can fail to match the input marriage rates. Often in such cases, it is necessary to "tune" the input rates slightly upward in order for the achieved to match those desired.

### 9.1 Marriage mechanics

The process whereby new new marriage market entrants choose or reject potential spouses from the *marriage queue* has two stages. In the first stage a *working marriage queue* is created for ego. This queue ultimately includes only those potential spouses who are both **eligible** and who have the highest "score" as determined by a `score()` function whose properties may be affected by the identity of the choosing suitor. Possibly this *working marriage queue* contains only a single highest scoring potential spouse. If it has more than one member, then they are all both eligible and equal in the eyes of `score()`. It is also possible for the *working marriage queue* to be empty.

---

<sup>9</sup>At present same sex marriage is not implemented in SOCSIM

<sup>10</sup>As noted below, since the `score()` and `pref()` functions can be set individually, it is possible to add this sort of effect to SOCSIM but it would require effort

In stage one, that is in the creation of the *working marriage queue*, the default version of SOCSIM considers eligible any potential spouse who:

- is **not** a first cousin or closer relative.
- is not a previous spouse.
- is of an age such that *groom's age - bride's age* is between `marriage_age_diff_min` and `marriage_age_diff_max`. These latter values are set in the `.sup` file.
- is not contraindicated by the setting of the `endogamy` parameter, as described in Section 5.

As the list is constructed, the `score()` function is called for each potential pair of spouses. In the default version this function assigns a value that is higher the closer the *groom's age-bride's age* is to the `marriage_peak_age`. The value declines with distance according to the value of `marriage_slope_ratio`. When `marriage_slope_ratio` is 1, `score` is indifferent between a marriages with equal absolute values of *groom's age-bride's age*. With `marriage_slope_ratio` set to 2, the decline in preference when the potential bride is older is twice as rapid as when the potential groom is older. See Section 7 for instructions on how to set the parameters related to admissible spousal age differences.

In the second stage, *after* the *working marriage queue* is constructed, one potential spouse from it is selected according to the `pref()` function. The `pref()` like the `score()` function, `pref()` function can be affected by the characteristics of the choosing spouse, however, unlike the `score()` function, `pref()` must not allow ties. Since all members of the *working marriage queue* are equally eligible and have equal “scores”, `pref()` is in effect the tie breaker. In the default version of SOCSIM, `pref()` simply chooses at random from the *working marriage queue*.

Both the `pref()` and `score()` can be modified or replaced, there could even be distinct `pref()` and `score()` functions for each individual. While the code is designed to make these changes as easy as possible, changing these functions does involve programming in C.