The RiskPaths teaching model
A simple demographic competing risk microsimulation model implemented in Modgen

Martin Spielauer

Statistics Canada - Modeling Division
R.H. Coats Building, 24-O
Ottawa, K1A 0T6

martin.spielauer@statcan.gc.ca

Paper presented at the Conference of the International Microsimulation Association

June 2009, Ottawa, Canada

Modgen is a generic microsimulation programming language developed and maintained at Statistics Canada. RiskPaths as well as the Modgen programming language and other related documents are available at www.statcan.gc.ca/microsimulation/modgen/modgen-eng.htm

Abstract

RiskPaths is a simple, competing risk, continuous time microsimulation model developed alongside a microsimulation course for the European Doctoral School for Demography. It enables the study of how childlessness and other measures are affected at an aggregate level by changes in individual processes, such as fertility by age, first and second union formation, and union dissolution. While kept simple, RiskPaths nevertheless demonstrates what microsimulation can add to event history analysis and how demographic microsimulation models can be efficiently programmed using the language Modgen. The output of RiskPaths includes the visual display of how individual risks of the modeled events change over the simulated life courses, thus allowing students to better understand the underlying hazard regression models and the concept of competing risks. Table output includes the simulated values of the initial model parameters, a feature which supports the study of randomness and its connection to sample size.

1 Introduction

This paper describes the RiskPaths teaching model developed to introduce dynamic microsimulation and the generic microsimulation programming language Modgen. It is organized in two main parts. We first present the RiskPaths model from the user's view; the second part focuses on the RiskPaths code.

The first part starts with a description of the underlying statistical models and then explores follow-up questions, such as what microsimulation can add to the initial statistical analysis and what other benefits microsimulation can bring to the overall analysis. We then demonstrate parts of Modgen's visual interface to examine elements of the RiskPaths model. RiskPaths can be used as a model to study childlessness and was developed for training purposes. Technically, RiskPaths is a demographic single sex (female only), data-driven, specialized, continuous time, case-based, competing risk cohort model. It is based on a set of piecewise constant hazard regression models.

In essence, RiskPaths allows the comparison of basic demographic behaviour before and after the political and economic transitions experienced by Russia and Bulgaria around 1989. Its parameters were estimated from Russian and Bulgarian data of the Generations and Gender Survey conducted around 2003/04. Russia and Bulgaria comprise interesting study cases since both countries, after the collapse of socialism, underwent the biggest fertility declines ever observed in history during periods of peace. Furthermore, demographic patterns were very similar and stable in socialist times for both countries, which helps to justify the use of single cohorts as a means of comparison (one representing life in socialist times, the other the life of a post-transition cohort). In this way, the model allows us to compare demographic behaviour before and after the transition, as well as between the two countries themselves.

In the second part we explore the microsimulation model development package Modgen and the Modgen application RiskPaths from the model developer's point of view. We first introduce the Modgen programming environment, and then discuss basic Modgen language concepts and the RiskPaths code. Modgen is a microsimulation model development package developed by and distributed through Statistics Canada. It was designed to ease the creation, maintenance, and documentation of microsimulation models without the need for advanced programming skills as a prerequisite. This is possible because Modgen hides underlying mechanisms like event queuing and automatically creates a stand-alone model with a complete visual interface, including scenario management and model documentation. Model developers can therefore concentrate on model specific code: the declaration of parameters, the states defining the simulated actors, and the events changing the states. High efficiency coding extends also to model output. Modgen includes a powerful language to handle continuous time tabulation. These tabulations are created on-the-fly when simulations are run and the programming to generate them usually requires only a few lines of code per table. Modgen also has a built-in mechanism for estimating the Monte

Carlo variation for any cell of any table, without requiring any programming by the model developer.

Being a simple model, RiskPaths does not make use of the full range of available Modgen language concepts and capabilities. The discussion in this paper does not intend to replace existing Modgen documentation, such as the Modgen Developer's Guide, either. But by introducing the main concepts of Modgen programming, we aim to help you get started in Modgen model development and to engage in further exploration.

Besides for teaching purposes, RiskPaths so far was also successfully used as template for two demographic models, developed for the study of the effect of union dissolution on fertility in France (Thomson et. al. 2006) and Canada (Bélanger et. al. 2006).

Part I: RiskPaths from the User's perspective

2 RiskPaths: The underlying statistical models

2.1 General description

Being a model for the study of childlessness, the main event of RiskPaths is the first pregnancy (which is always assumed to lead to birth). Pregnancy can occur at any point in time after the 15th birthday, with risks changing by both age and union status. The underlying statistical models are piecewise constant hazard regressions. With respect to fertility this implies the assumption of a constant pregnancy risk for a given age group (e.g. age 15-17.5) and union status (e.g. single with no prior unions).

For unions, we distinguish four possible state levels:

- single
- the first three years in a first union
- the following years in a first union
- all the years in a second union

(After the dissolution of a second union, women are assumed to stay single). Accordingly, we model five different union events:

- first union formation
- first union dissolution
- second union formation
- second union dissolution
- the change of union phase which occurs after three years in the first union.

The last event (change of union phase) is a clock event – it differs from other events in that its timing is not stochastic but predefined. (Another clock event in the model is the change of the age index every 2.5 years) Besides unions and fertility, we model mortality--a woman may die at any point in time. We stop the simulation of the pregnancy and union events either when a women dies, or at pregnancy (as we are only interested in studying childlessness), or at her 40th birthday (since later first pregnancies are very rare in Russia and Bulgaria and are thus ignored for this model).

At age fifteen a woman becomes subject to both pregnancy and union formation risks. These are competing risks. We draw random durations to first pregnancy and to first union formation. There are two additional competing events at this stage—mortality and change of age group. (As we assume that both pregnancy and union formation risks change with age, the risks underlying the random durations only apply for a given time period--2.5 years in our model--and have to be recalculated at that point in time.)

In other words, the 15th birthday will be followed by one of these four possible events:

- the woman dies
- she gets pregnant
- she enters a union
- she enters a new age group at age 17.5 because none of the first three events occurred before age 17.5

Death or pregnancy terminates the simulation. A change of age index requires that the waiting times for the competing events union formation and pregnancy be updated. The union formation event alters the risk of first pregnancy (making it much higher) and changes the set of competing risks. A woman is then no longer at risk of first union formation but becomes subject to union dissolution risk.

2.2 Events and parameter estimates

2.2.1 First pregnancy

As outlined above, first pregnancy is modeled by an age baseline hazard and relative risks dependent on union status and duration. The following Table 1 displays the parameter estimates for Bulgaria and Russia before and after the political and economical transition.

Table 1: First pregnancy risks

	Bulgaria	Russia
15-17.5	0.2869	0.2120
17.5-20	0.7591	0.7606
20-22.5	0.8458	0.8295
22.5-25	0.8167	0.6505
25-27.5	0.6727	0.5423
27.5-30	0.5105	0.5787
30-32.5	0.4882	0.4884
32.5-35	0.2562	0.3237
35-37.5	0.2597	0.3089
37.5-40	0.1542	0.0909

	before 1989 transition		10 years after transition: 1999+	
	Bulgaria	Russia	Bulgaria	Russia
Not in union	0.0648	0.0893	0.0316	0.0664
First 3 years of first union	1.0000	1.0000	0.4890	0.5067
First union after three years	0.2523	0.2767	0.2652	0.2746
Second union	0.8048	0.5250	0.2285	0.2698

The data from Table 1 is interpreted as follows in the model. As long as a woman has not entered a partnership, we have to multiply her age-dependent baseline risk of first pregnancy by the relative risk "not in a union". For example, the pregnancy risk of a 20 year old single woman of the pre-transition Bulgarian cohort can be calculated as 0.8458*0.0648 = 0.05481. At this rate of λ =0.05481:

- The expected mean waiting time to the pregnancy event is $1/\lambda = 1/0.05481 = 18.25$ years;
- The probability that a women does not experience pregnancy in the following 2.5 years (given that she stays single) is $\exp(-\lambda t) = \exp(-0.05481*2.5) = 87.2\%$.

Thus at her 20th birthday, we can draw a random duration to first pregnancy from a uniform distributed random number (a number that can obtain any value between 0 and 1 with the same probability) using the formula:

RandomDuration = -ln(RandomUniform) / λ;

As we have calculated above, in 87.2% of the cases, no conception will take place in the next 2.5 years. Accordingly, if we draw a uniform distributed random number smaller than 0.872, the corresponding waiting time will be longer than 2.5 years, since -ln(RandomUniform) / λ = -ln(0.872)/0.05481 = 2.5 years. A random draw greater than 0.872 will result in a waiting time smaller than 2.5 years–in this situation, if the woman does not enter a union before the pregnancy event, the pregnancy takes place in our simulation.

To continue this example, let us assume that the first event that happens in our simulation is a union formation at age 20.5. We now have to update the pregnancy risk. While the baseline risk still stays the same for the next two years (i.e. 0.8458), the relative risk is now 1.0000 (as per the reference category in Table 1) because the woman is in the first three years of a union. The new hazard rate for pregnancy (applicable for the next two years, until age 22.5) is considerably higher now at 0.8458*1.0000 = 0.8458. The average waiting time at this rate is thus only 1/0.8458 = 1.18 years and for any random number greater than exp(-0.8458*2)=0.1842 the simulated waiting time would be smaller than two years. That is, 81.6% (1 – 0.1842) of women will experience a first pregnancy within the first two years of a first union or partnership that begins at age 20.5.

2.2.2 First union formation

Risks are given as piecewise constant rates changing with age. Again we model age intervals of 2.5 years. These are the rates for women prior to any conception, as such an event would stop our simulation.

Table 2: First union formation risks

	before 1989 transition		10 years after transition: 1999+	
	Bulgaria	Russia	Bulgaria	Russia
15-17.5	0.0309	0.0297	0.0173	0.0303
17.5-20	0.1341	0.1342	0.0751	0.1369
20-22.5	0.1672	0.1889	0.0936	0.1926
22.5-25	0.1656	0.1724	0.0927	0.1758
25-27.5	0.1474	0.1208	0.0825	0.1232
27.5-30	0.1085	0.1086	0.0607	0.1108
30-32.5	0.0804	0.0838	0.0450	0.0855
32.5-35	0.0339	0.0862	0.0190	0.0879
35-37.5	0.0455	0.0388	0.0255	0.0396
37.5-40	0.0400	0.0324	0.0224	0.0330

The parameterization example given in Table 2 has the following interpretation: the first union formation hazard of Bulgarian women of the first cohort is 0 until the 15th birthday; afterwards it changes in time steps of 2.5 years from 0.0309 to 0.1341, then from 0.1341 to 0.1672, and so on. The risk is highest for the age group 20-22.5--at a rate of 0.1672, the expected time to union formation is 1/0.1672=6 years. A women who is single on her 20th birthday has a 34% probability of experiencing a first union formation in the following 2.5 years (p=1-exp(-0.1672*2.5)).

2.2.3 Second union formation

A woman becomes exposed to the second union formation risk if and when her first union dissolves. As a difference to the first union formation which is based on age, this process does

not start at a fixed point in time but is triggered by another event (first union dissolution). Accordingly, the time intervals of the estimated piecewise constant hazard rates refer to the time since first union dissolution.

Table 3: Second union formation risks

	before 1989 transition		10 years after transition: 1999+	
	Bulgaria	Russia	Bulgaria	Russia
<2 years after dissolution	0.1996	0.2554	0.1457	0.2247
2-6 years after dissolution	0.1353	0.1695	0.0988	0.1492
6-10 years after dissolution	0.1099	0.1354	0.0802	0.1191
10-15 years after dissolution	0.0261	0.1126	0.0191	0.0991
>5 years after dissolution	0.0457	0.0217	0.0334	0.0191

2.2.4 Union dissolution

Both first and second unions can dissolve, with such processes starting at the first and second union formations, respectively. As the sample size is very small for the modeling of the second union dissolution event we do not distinguish the before and after transition cohorts for this event.

Table 4: First union dissolution risks

	before 1989 transition		10 years after transition: 1999+	
	Bulgaria	Russia	Bulgaria	Russia
First year of union	0.0096	0.0380	0.0121	0.0601
Union duration 1-5	0.0200	0.0601	0.0252	0.0949
Union duration 5-9	0.0213	0.0476	0.0269	0.0752
Union duration 9-13	0.0151	0.0408	0.0190	0.0645
Union duration >13	0.0111	0.0282	0.0140	0.0445

Table 5: Second union dissolution risks

	Bulgaria	Russia
First 3 years of union	0.0371	0.0810
Union duration 3-9	0.0128	0.0744
Union duration 9+	0.0661	0.0632

2.2.5 Mortality

In this sample model, we leave it to the model user to either set death probabilities by age or to "switch off" mortality allowing the study of fertility without interference from mortality. In the latter case, all women reach the maximum age of 100 years. If the user chooses to simulate mortality, the specified probabilities are internally converted to piecewise constant hazard rates (based on the formula $-\ln(1-p)$ for $p<1$) so that death can happen at any time in a year. If a probability is set to 1 (as is the case when age=100), immediate death is assumed.

3 What do we expect from the microsimulation model RiskPaths?

3.1 What can simulation add to statistical analysis?

Before we can answer the question of what simulation can add to statistical analysis, we first need a good understanding of what the statistical results presented in the previous section reveal. The estimation results for the two countries and two cohorts allow us to study similarities and differences between the countries, as well as the changes in parameters over time *separately* for each of the individual processes. We see a remarkable similarity in parameters across the two countries especially for the pre-transition cohorts. Bulgaria differs from Russia basically only in the three times lower union dissolution risks and the slower speed of second union formation. Accordingly, comparing the pre- and post-transition cohorts, we find dramatic changes in most processes. The risk of first births was halved in the first three years of the first union with no later recovery, although the parameters stayed relatively unchanged after three years in a union. Also, in second unions, fertility dropped by more than 50%. The biggest difference between the two countries after the transition is in first union formation--rates halved in Bulgaria but stayed stable in Russia. For first union dissolution we see the opposite picture--union dissolution risks increased by around 40% in Russia while staying almost unchanged in Bulgaria.

These are typical examples of insights we can gain by *single process analysis*. We have separated a complex system into its component processes and studied the changes within those processes. In the case of fertility we have introduced relative risks--we study how certain factors (here, different union statuses) influence a *single* process. This is a very typical analytical question; scientific literature is rich of this kind of research.

The power of microsimulation unfolds when we study various processes simultaneously. Even in our very simple demographic example, results are difficult to interpret when we are interested in the effect of changes in single processes on aggregate outcomes. For example, what is the effect of Russia's 40% increase in union dissolution risks on childlessness? The effect will depend on fertility out of unions and in second unions as well as the speed of second union formation. The relative risk of fertility is higher in second unions than after three years in the first union, but second union formation takes time (during which fertility is very low) and not all women enter a second union. Do these effects cancel themselves out or does union dissolution affect fertility - and in which direction? Such questions invite us to use microsimulation for *sensitivity analysis*. How do aggregate outcomes change in response to the change of a single parameter? Note that we now have moved analysis from the level of a single process to an *analysis of system behaviour*.

A comparison of the two cohorts invites a further type of system analysis--what is the relative contribution of the change in single processes to the aggregated outcome? Comparing the two simulated cohorts we see that childlessness has increased considerably in both countries but even more so in Bulgaria. We can use microsimulation to *decompose* the contributions of the changes

in the various processes to the aggregate change. How much would childlessness have changed if only fertility parameters changed? What is the contribution of changes in union formation? Has the increase in union dissolution risk contributed to the increase in childlessness in Russia? Of course, the aggregate change is not the simple arithmetic sum of partial effects. Some process changes might have a stronger or weaker effect in the presence of changes in other processes. For example, the effect of the change in fertility in second unions will heavily depend on the likelihood of being in a second union which is subject to first union formation and dissolution risks. Microsimulation can help us to identify and better understand such interactions.

Looking at the post-transition cohort, we have already entered the domain of *predictions*. As data were collected 14 years after the transition, in reality no post-transition cohort has gone through its whole reproductive period. Thus, for cohort measures like childlessness, the assessment of consistency with other data sources is limited to a comparison with other projections. But we can also use our model for predictions under alternative assumptions on future changes in processes. We might have a theory that leads to the assumption that only parts of the observed changes are of a permanent nature (e.g. caused by cultural change) while others are transitory (e.g. resulting from economic crisis, therefore reversible with economic recovery). What would happen if fertility rates moved back to their initial values while slower (later) union formation persisted--or vice versa? Such an analysis can produce surprising results, as it is not always a reversal of the process which initially had the biggest overall impact that will generate the biggest opposite effect.

3.2 Desired features of a RiskPaths microsimulation model

3.2.1 Input: Parameter tables, scenarios, and simulation settings

Even being a very simple model, RiskPaths has around 130 parameter values which users should be able to set and store conveniently. We would expect these parameters to be well-organized in the microsimulation application, appearing as easy-to-access (or navigate) labelled tables which could be read or modified as required

When using a model we typically create different scenarios, i.e. different parameterizations of the model. We need to be able to save these scenarios so that certain simulations can be reproduced in future. Scenarios contain all parameter tables and, ideally, supplementary text descriptions or notes that outline the specific changes embedded in each scenario. Additionally, scenarios should include scenario settings, such as the number of simulated cases (given that RiskPaths is a case-based model), A large sample size will reduce Monte Carlo variation but comes at the cost of slower simulation runs. If we are only interested in broad aggregates, then smaller sample sizes might suffice. On the other hand, a detailed analysis of rare events or a detailed breakdowns of results (e.g. by age groups) would require large samples. Additionally, users might not wish to

produce all available output. Narrowing down the desired output can again speed up simulations but also leads to a more concise and focused presentation of results according to user needs.

All of the above (parameter tables, descriptive notes, number of cases, choice of output to produce) is part of a scenario. For our RiskPaths applications, we would expect all this information to be stored together for a given scenario and we would expect it to be easily retrieved, viewed, and modified.

3.2.2 Output and output views

Microsimulation models can produce output on two levels: micro and macro. A microsimulation application could conceivably write all individual level characteristics and all their changes over time into a file and leave it to the user to analyse the resulting data file with statistical software. In the RiskPaths case, this would lead to a file storing the dates of all simulated events that occur over the simulated life course of each single individual. Only six events can happen in a simulated life, so each data record would contain at most six variables: four union formation / dissolution events, conception, and death. For more complex applications, file size and complexity could be enormous.

As well as such a longitudinal file, we might also be interested in cross-sectional output, recording the states of all individuals at a certain point in time. While the use of such a file is rather limited when simulating a single cohort, it would resemble a cross-sectional survey or population census in a population model.

Usually, a model user will not be interested in micro files per se but in the analysis that is performed on them. The user will typically aggregate data and produce summary indicators and tables. If model developers already know how simulated data will or should be analyzed, such measures and tables can already be calculated and produced within a microsimulation application run. In this case, users would not need to run additional statistical routines; they could see results immediately after a simulation was performed. In our RiskPaths model, output does not exceed a small number of tables and summary indicators which we expect to be produced within the application. We are interested in age-specific fertility rates, childlessness, the mean age at first conception, first conception by union status, and some mortality measures.

Just as with parameter tables, aggregated model output also requires organization. We might want to present some summary measures of one or several related behaviours together in a table and we surely want to order table output in a meaningful way. Additionally, as with parameters, we would expect table results to be labelled for easy reading and understanding.

Because all microsimulation results are subject to Monte Carlo variation, aggregated numbers are only one view of the results. We might also be interested in getting distributional information on each table value. Such information would help us to set an appropriate population size sufficient for a desired level of result precision.

A special type of micro-data output is the graphical display of individual careers. This can be a helpful feature, as it provides users with a window to the simulated individuals, and thus a way to see the operation of the statistical models. This can also be useful for model developers as it supports model debugging. Since RiskPaths is a training tool, we are interested in displaying how individual biographies result from statistical processes. Thus, besides life course events, we might also want to see how the risks of the alternative events change over time and life course situations.

3.2.3 User interface and documentation

So far, we have formed expectations about the content, display, and organization of model input and output data. From the user perspective, do we just have to add a start button to complete the microsimulation application? Almost all contemporary software applications contain help files. As users of microsimulation models, we should expect access to detailed online help, not only on the use of the modeling software itself but also on the model's specific elements and the interrelationships amongst those elements.

4 Exploring the Modgen application RiskPaths

In the remainder of this discussion we provide a quick explorative tour of the visual interface provided by Modgen for the RiskPaths model. To run RiskPaths, both the Modgen Prerequisites application and the RiskPaths executable have to be installed on your computer. As is true for all Modgen applications, RiskPaths contains a help system including documentation on the (model-independent) Modgen user interface and the actual RiskPaths model itself. Accordingly, in the description below, we concentrate on the central steps of running RiskPaths, leaving it to you to explore the model and software in depth with the assistance of the detailed help files.

4.1 The user interface

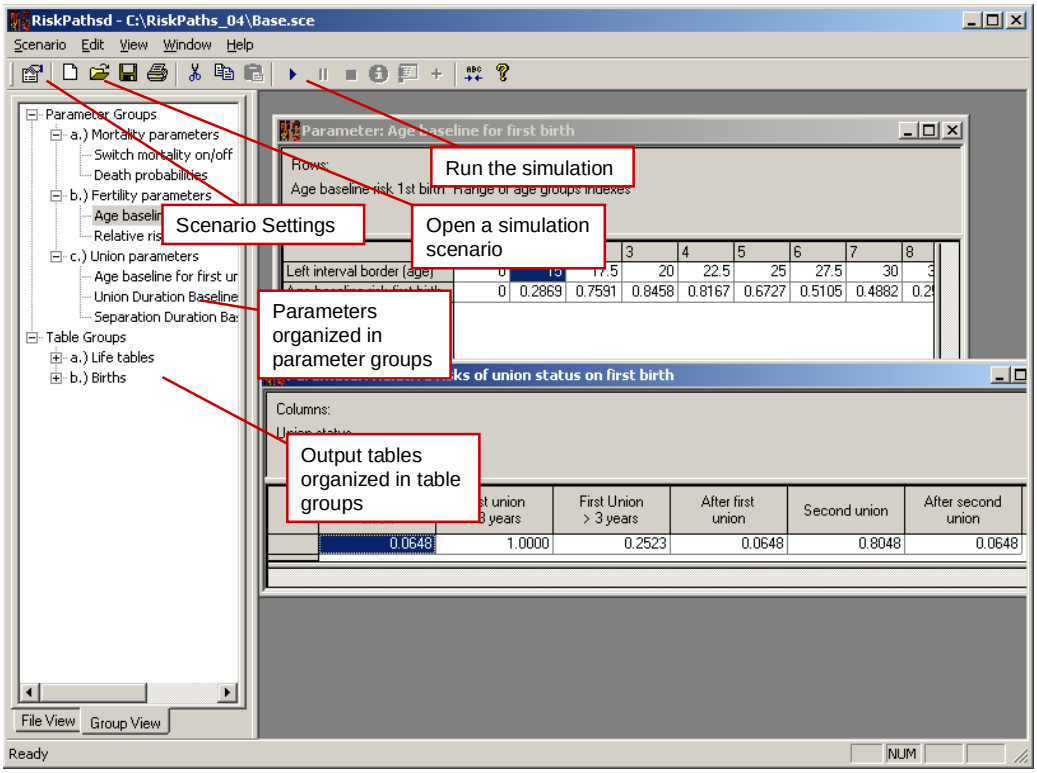
All Modgen applications have the same graphical user interface (Figure 1) which consists of the following parts:

- a menu bar and a toolbar to administer and run scenarios, as well as to get help
- a selection window containing a hierarchically grouped list of all model parameters and output tables
- a frame in which all corresponding parameters or tables can be displayed

When starting the RiskPaths.exe application, the selection window and table frame are empty, as we first have to load (or create) a simulation scenario. To do so, follow the following steps:

- Open the simulation scenario ‘base.sce’. This can be done by clicking the ‘Open’ button or by selecting ‘Open…’ from the ‘Scenario’ menu.
- Choose the scenario settings--the settings dialog box can be accessed by clicking the ‘Settings’ button or by selecting ‘Settings…’ from the ‘Scenario’ menu. Specify a small number of simulated cases (e.g. 10,000) so that your first model runs quickly. Also, ensure that ‘MS Access tracking’ is switched on. This will allow you to view individual biographies using the BioBrowser tool that comes with Modgen
- Save your scenario under a new name by selecting ‘Save as…’ from the ‘Scenario’ menu.

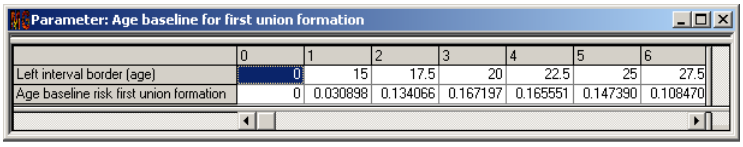
Figure 1: The RiskPaths application



4.2 Parameter tables

Users of a Modgen application have control over all parameters contained in the model’s parameter tables. An individual parameter table can be selected by clicking its list entry in the selection window. The table is then displayed in the display frame in which it can also be edited. Modgen parameter tables can have any number of dimensions, ranging from a parameter with a single checkbox to parameters with numerous characteristics or dimensions (e.g. region, sex, age, time).

Figure 2: Parameterization of first union formation risks



4.3 Performing a simulation run

Click the 'Run/resume' button or select 'Run/resume' from the 'Scenario' menu. The progress of the simulation is displayed in a progress dialog box. A small sample of 10,000 actors takes around 20 seconds to run. After the model run is complete, all output tables will have been updated by Modgen.

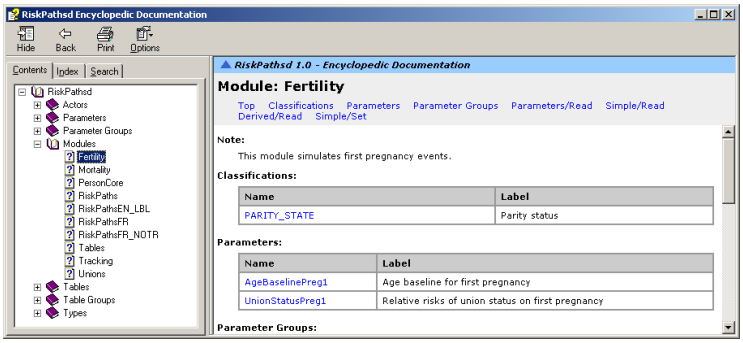
4.4 Table output: aggregates and distributions

Simulation results are written to predefined output tables. Note that the values displayed in the output table represent only one of several possible views on the results. By right-clicking a table, a properties sheet for the table can be accessed. Among other things, this allows the display of distributional information (standard errors and the coefficient of variation) of all simulated values. Table contents can also be copied and pasted. You have the choice to copy the table as displayed, or all dimensions of the table at once (if there are more than two dimensions).

4.5 Model help and documentation

As is true with all Modgen applications, RiskPaths provides help files of various types. Two are related to Modgen itself--a general user guide for the visual interface plus release notes for Modgen. The other help files are model-specific. All Modgen applications contain a detailed encyclopaedic model documentation file. This documentation is automatically created from properly commented code.

Figure 3: Model documentation

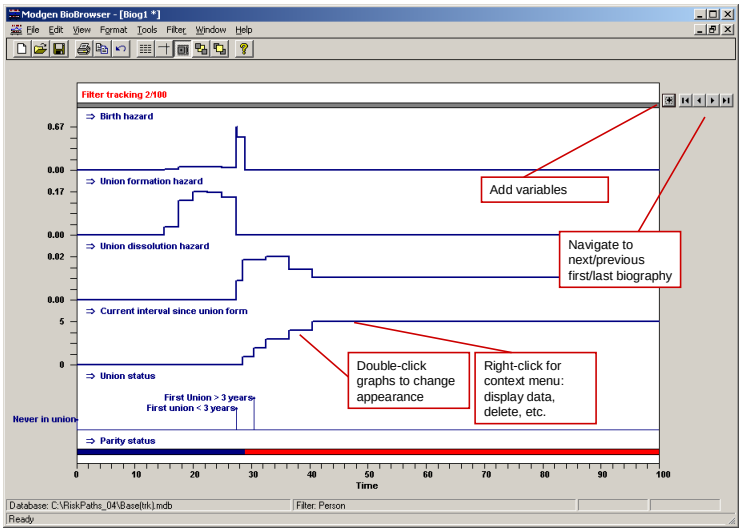


4.6 Graphical output of individual histories

The Modgen Biography Browser (BioBrowser) application is a tool for the graphical display of individual life courses. This view on the simulation results is especially useful for model debugging. In order to use the tool, the tracking feature has to be switched on in the scenario settings. The list of variables to be tracked also has to be declared by the model developer in the model code via a tracking statement. Modgen than tracks all changes of those variables included in the tracking statement for a sample of simulated actors (where the size of this sample is specified as one of the scenario settings).

To display biographies created by RiskPaths, just start the BioBrowser application and load the tracking-file of your simulation scenario, e.g. Base(trk).mdb.

Figure 4: BioBrowser



Part II: The RiskPaths Code

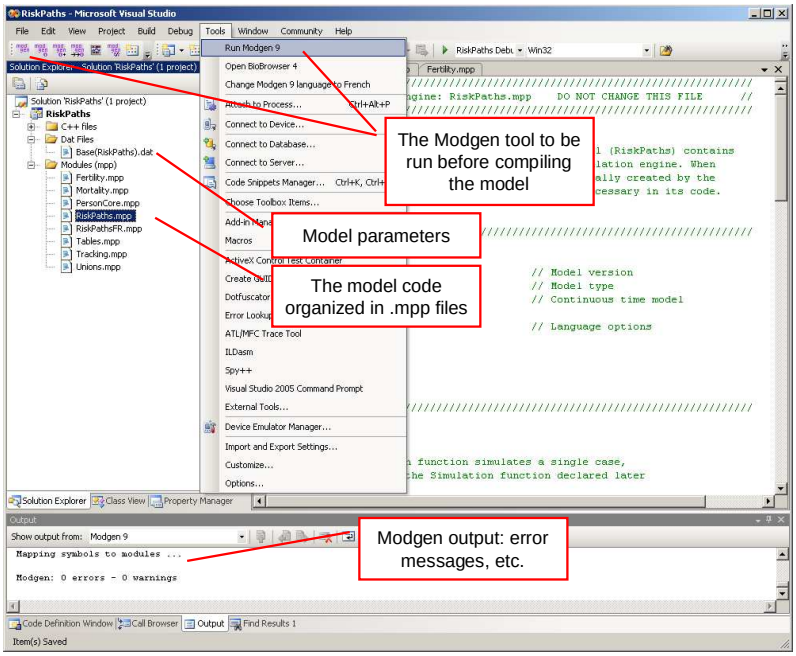
5 The Modgen programming environment

When installed on a computer, Modgen integrates itself into the (required) Microsoft Visual Studio C++ environment. The visual components of Modgen are a separate toolbar as well as additional items under the Tools and Help menus of Visual Studio. Modgen also appears as an option in the file dialog box for creating a new project as well as in the dialog box for adding a file to an existing project.

Figure 5 displays a screenshot of the programming interface as it appears after opening the Modgen application 'RiskPaths.sln'. The Modgen toolbar consists of several icons for running Modgen, accessing help, opening the BioBrowser tool, and switching the language (between English and French).

Modgen code is organized into several files, each with the file extension .mpp. As can be seen in the Solution Explorer window (Figure 5), RiskPaths consists of eight .mpp files grouped in the "Models (mpp)" folder. These are the essential files of RiskPaths, i.e. the files containing all Modgen code written by the model developer.

Figure 5: The programming interface



When invoking the Modgen tool (which can be accessed from the toolbar, or from the first item under the "Tools" menu), these .mpp files are translated into C++ code. Thus Modgen acts as a pre-compiler, creating one .cpp source code file for each .mpp file and putting the resulting .cpp files in the "C++ Files" folder. The Modgen tool also adds model-independent C++ code components to the "C++ Files" folder; these additional files[1] should not be changed by the model developer and are essential in order to use the C++ compiler to build the Modgen application.

The model parameters are contained in one or more .dat files organized in a folder labelled "Scenarios". These files are loaded at runtime and contain the actual values assigned to the parameters.

When running the Modgen tool, Modgen – like the C++ compiler - produces log output that is displayed in the Output window. Any error messages are also displayed in this window, and clicking on a particular error message leads you directly to the corresponding Modgen code that produced the error.

[1] ACTORS.CPP, ACTORS.H, app.ico, model.h, model.RC, PARSE.INF, TABINIT.CPP, TABINIT.H.

Two steps are required to create a Modgen application from the Visual Studio environment. First, Modgen has to translate the Modgen code in the .mpp files; this is done when invoking the Modgen tool. Second, the resulting C++ application has to be built and started. This can be done in one step by selecting "Start Debugging" in the "Debug" menu or by clicking the corresponding icon at the toolbar.

6 Basic Modgen Concepts

Actor: An actor is the entity whose life is simulated in a Modgen model. A model's actor is often a person, although this is not a requirement--other models have been developed that use dwellings or occupations as actors. Nevertheless, in RiskPaths, the actor is a person or more specifically, a female (since it is a model for the study of childlessness)

State: States describe the characteristics of a model's actors. Some states can be continuous, such as age, whereas others are categorical, such as gender. For categorical states, the actual categories or levels are defined via Modgen's **classification** command.

Overall, there are two major kinds of states in Modgen—**simple states** and **derived states**, both of which are used by RiskPaths and both of which are declared within an actor declaration. A simple state is a state whose value can be initialized and changed by the code that a model developer creates. Simple states are changed by explicitly declared events. A derived state, on the other hand, is a state whose value is given as an expression which is normally derived from or based on other states. A derived state's values are automatically maintained by Modgen throughout a simulation run. A useful Modgen concept is the self-scheduling derived state. This is a state which changes in a predefined time sequence, such as integer age which will change at each birthday.

Event: In Modgen, simulation takes place through the execution of events. Each event consists of two functions: a time function to determine the time of the next occurrence of the event, and an implementation function to determine the consequences when the event happens. RiskPaths has several events, including a mortality event, union formation and dissolution events, and a first pregnancy event.

Parameter: Parameters are used to give model users a degree of control over the simulations they run. The ability to change different hazards or probabilities that affect various aspects of a simulation allows different scenarios to be explored. Parameters can have many dimensions (such as age, gender, and year) and are stored in .dat data files. In RiskPaths, there is one parameter file, Base(RiskPaths).dat, which stores parameters such as death probabilities by age and risks of first pregnancy by age group. More complex models will usually incorporate more than one .dat file.

Table: Modgen has a powerful cross-tabulation facility built in to report aggregated results in the form of tables. There are two central elements of a table declaration—its captured dimensions

(defining when an actor enters and leaves a cell) and its analysis dimension (recording what happens while an actor is in that cell). When running simulations, the tabulations to fill a table are created on the fly, thus removing the need to create and write to large temporary interim files for subsequent reporting. Several examples of table declarations will be shown later in this chapter for RiskPaths.

7 Organization of files

The Modgen code of RiskPaths is organized into eight separate .mpp files, while all RiskPaths parameter values (because RiskPaths is a simple model) are contained in just a single .dat file. In principle, a model developer has complete freedom to decide how to organize the Modgen code in different files, but a modular organization as found in RiskPaths is recommended.

Figure 6: RiskPaths file organization

General modules	Filename
Simulation engine	RiskPaths.mpp
Core actor file	PersonCore.mpp
Table definitions	Tables.mpp
Output tracking	Tracking.mpp
French language translations	RiskPathsFR.mpp

Behavioural modules	Filename
Mortality	Mortality.mpp
Fertility	Fertility.mpp
Union formations and dissolutions	Unions.mpp

Parameter file	Filename
Parameters of Baseline Scenario	Base(RiskPaths).dat

Note that the .mpp code files often contain comments that resemble labels. Such comments are placed beside the declarations of symbols such as states, state levels, parameters, tables and table dimensions. Modgen does in fact interpret these comments as labels and subsequently uses them when tables or parameters are displayed within Modgen's visual interface. These labels are also used in the model's automatically generated encyclopaedic help file. Code comments that are used as labels begin with a two-character language identifier, e.g. //EN Union status. Many such comments can be seen in the code examples that follow for RiskPaths.

Fore more detailed descriptions of modules, functions and events, notes following the following syntax example can be placed in the code. These notes – besides documenting the code - are used in the automatically generated encyclopaedic help file.

```
/*NOTE(Person.Finish, EN)
      The Finish function terminates the simulation of an actor.
*/
```

7.1 RiskPaths.mpp (the main simulation file)

This file contains the code essential for the definition of the model type (e.g. case-based, continuous time) as well as the simulation engine, i.e. the code that runs the whole simulation. Because RiskPaths is a case-based model, the simulation engine code loops through all cases and processes the event queues of each case. The file also identifies the languages of the model. The code of this file is mostly model independent within a class of models (e.g. continuous time, case-based) and a version of it is provided automatically when using Modgen's built-in wizards to start a new Modgen project.

For the development of our case-based, continuous time cohort RiskPaths model with an actor 'Person' the code provided by the wizard requires very few modifications. The full code of this .mpp file is less than one page in length.

7.2 PersonCore.mpp

The only actor in RiskPaths is a person. In the file PersonCore.mpp, we have organized the code which is part of the actor declaration but not directly related to a specific behaviour. The file contains two age clocks defined as self-scheduling states (integer_age and age_status) and two actor functions, Start() and Finish(), which are performed at the creation of an actor and at her death, respectively.

In the Start() function we initialize the states time and age to 0. Both states are automatically created and maintained by Modgen and can only be changed in the Start() function. Their types depend on the model type; because RiskPaths is a continuous time model, time and age are continuous states.

The Finish() function must be called at the death event of an actor. Its role is to remove the actor from tables and from the simulation, and to recuperate any computer memory used by the actor.

All states and actor functions are declared in an "actor Person { };" block. To allow modularity in the organization of code by different life course domains, there can be multiple actor blocks in a project, typically one for each behavioural file.

The first code section of this module contains three type definitions. We first define a range LIFE.

```
range LIFE        //EN Simulated age range
{
      0,100
};
```

Range is a Modgen type which defines a range of integer values. RiskPaths limits the possible age range of persons to 100 years. This type will be used to declare a derived state containing the age of a person in completed years. The second type definition will be used to divide continuous age into 2.5 year age intervals starting at age 15.

```
partition AGEINT_STATE        //EN 2.5 year age intervals
{
      15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40
};
```

The third definition is a classification of union types. In general, if a range, partition, or classification is used in several files, it is good practice to define it in the core actor file.

```
classification UNION_STATE                //EN Union status
{
      US_NEVER_IN_UNION,                  //EN Never in union
      US_FIRST_UNION_PERIOD1,             //EN First union < 3 years
      US_FIRST_UNION_PERIOD2,             //EN First Union > 3 years
      US_AFTER_FIRST_UNION,               //EN After first union
      US_SECOND_UNION,                    //EN Second union
      US_AFTER_SECOND_UNION               //EN After second union
};
```

In the following code segment we declare two derived actor states and two functions. The derived states for time intervals are used to change the values of parameters that vary over time. In our model integer_age is needed because mortality risks are dependent on age in years, whereas age_status comes into play because baseline risks for first conception and first union formation are modelled to change in 2.5 year intervals after the 15th birthday. Both integer_age and age_status have to be maintained over the simulation. The Modgen concept of derived states allows us to have them maintained automatically. Both are derived from the state age (which is a special state, as it is generated and maintained automatically by Modgen). In order to split up age into the time intervals defined in the AGEINT_STATE partition, we make use of the Modgen function self_scheduling_split. The second derived state, integer_age, could be directly obtained using the Modgen function self_scheduling_int. In order to ensure that its value stays in the possible range of LIFE we convert it to type LIFE, which is done by the Modgen macro COERCE.

```
actor Person
{
      //EN Current age interval
      int age_status = self_scheduling_split(age, AGEINT_STATE);

      //EN Current integer age
      LIFE integer_age = COERCE( LIFE, self_scheduling_int(age) );

      //EN Function starting the life of an actor
      void Start();

      //EN Function finishing the life of an actor
      void Finish();
}
```

The remaining code of this module is the implementation of the Start() and Finish() functions. The Finish() function is left empty as we do not require any actions, other than those automatically performed by Modgen, to take place when an actor dies.

```
void Person::Start()
{
      // Age and time are variables automatically maintained by
      // Modgen. They can be set only in the Start function
      age = 0;
      time = 0;
}

/*NOTE(Person.Finish, EN)
      The Finish function terminates the simulation of an actor.
*/
void Person::Finish()
{
      // After the code in this function (if any) is executed,
      // Modgen removes the actor from tables and from the simulation.
      // Modgen also recuperates any memory used by the actor.

}
```

7.3 Behavioural Files

In RiskPaths we distinguish three groups of behaviours: mortality, fertility and union formation/dissolution. Accordingly we have organized the code into three .mpp files: Mortality.mpp, Fertility.mpp and Unions.mpp. Behavioural files are typically arranged in three sections:

- Declaration of parameters
- Declarations of actor states and events
- Implementation of events

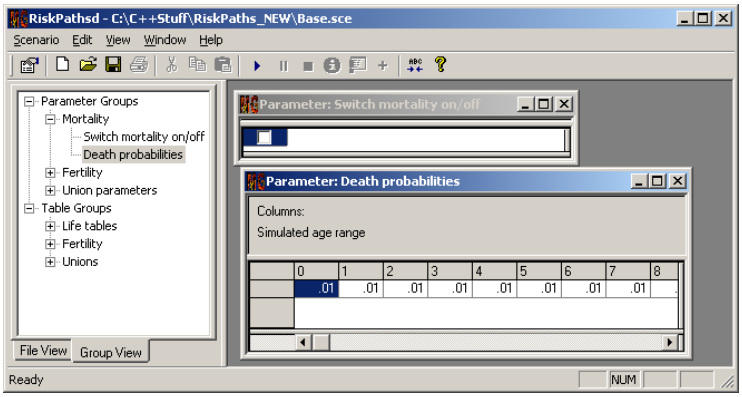
7.3.1 Mortality.mpp

This file defines the mortality event that ends the life of the simulated actor. Mortality.mpp is a typical behavioural module, and we follow a standard organization of the code: type definitions, parameter declarations, actor declarations and event implementations.

Parameter declarations

Mortality is parameterized by death probabilities by age; thus, the probability to survive another year changes at each birthday. We also introduce a parameter which allows us to 'switch off' mortality. When it is used, every actor reaches the maximum age of 100 years (which can be useful for some types of fertility analysis). Figure 7 displays the mortality parameter tables of the RiskPaths application.

Figure 7: Mortality parameters



Parameters are declared within a "parameters {…};" code block. Modgen supports numeric types, such as int, long, float, double, or Boolean ("logical" in the Modgen terminology). The dimensionality of the parameters in the RiskPaths model is defined by classifications and ranges. The following code generates the parameters for RiskPaths, as displayed in Figure 7. For the annual death probabilities we use the range LIFE that was defined in PersonCore.mpp. The following statement (**parameter_group)** groups the two mortality parameters in order to provide an ordered hierarchical selection list in the user interface (again, as displayed in Figure 3).

```
parameters
{
      logical CanDie;                //EN Switch mortality on/off
      double  ProbMort[LIFE];        //EN Death probabilities
};

parameter_group P01_Mortality        //EN Mortality
{
      CanDie, ProbMort
};
```

Actor declarations

Actors are described by states which are changed in events. States can be both continuous (integer or real) or categorical. In the mortality module, the state of interest is whether a person is alive or not, thus making it categorical in nature. The levels of a categorical state are defined with the Modgen **classification** command.

We declare a state life_status of type LIFE_STATE, which is initialized with LS_ALIVE at birth and set to LS_NOT_ALIVE by the death event. It is good practice to initialize all states by assigning initial values. Each initial value, however, must be enclosed in braces, i.e. {}—otherwise, the state is implemented as a derived state.

```
classification LIFE_STATE                          //EN Life status
{
      LS_ALIVE,                                    //EN Alive
      LS_NOT_ALIVE                                 //EN Dead
};

actor Person
{
      LIFE_STATE life_status = {LS_ALIVE};         //EN Life Status
      event timeDeathEvent, DeathEvent;            //EN Death Event
};
```

Events are declared in the actor Person {..} block using the keyword **event**. All events consist of a function which returns the time of the next event and a function containing the code describing the consequences of the event.

Event implementation

When mortality is activated, the timeDeathEvent function returns a random time based on the mortality parameter for the given year of age. In order to obtain random durations from probabilities, we assume constant mortality hazards within each period, i.e. between birthdays. (The exception is a death probability of 1, which leads to death immediately at the start of the age year). Note that any time later than the next birthday will lead to the birthday event taking precedence over the mortality event; that is, the birthday event will censor the mortality event.

```
TIME Person::timeDeathEvent()
{
      TIME event_time = TIME_INFINITE;
      if (CanDie)
      {
            if (ProbMort[integer_age] >= 1)
            {
                  event_time = WAIT(0);
            }
            else
            {
                  event_time = WAIT(-log(RandUniform(3)) /
                                    -log(1 - ProbMort[integer_age]));
            }
      }
      // Death event can not occur after the maximum duration of life
      if (event_time > MAX(LIFE))
      {
            event_time = MAX(LIFE);
      }
      return event_time;
}
```

The event implementation function DeathEvent is straightforward. It sets the life_status to LS_NOT_ALIVE and calls the function Finish(), the latter which deletes the actor.

```
void Person::DeathEvent()
{
	life_status = LS_NOT_ALIVE;
	Finish();
}
```

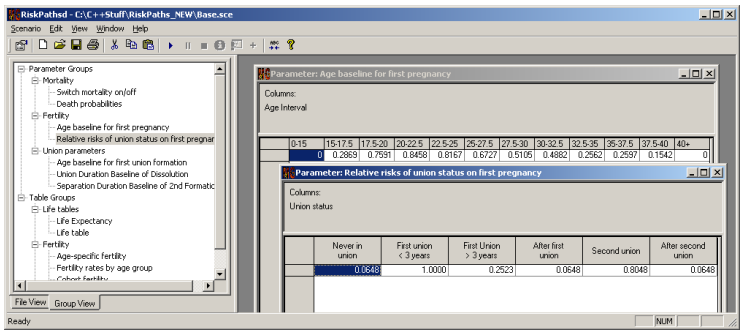
7.3.2 Fertility.mpp

This file defines and implements the first pregnancy event. As we are only interested in the study of childlessness in RiskPaths, no other fertility-related event is simulated. Fertility.mpp is a behavioural module, and again we follow the same standard organization of the code: type definitions, parameter declarations, actor declarations and event implementations.

Parameter declarations

Fertility is parameterized by both a baseline pregnancy risk by 2.5 year age intervals starting at the 15th birthday and a relative risk factor dependent on the union status and duration. We thus define two parameters: AgeBaselinePreg1 and UnionStatusPreg1.

Figure 8: Fertility parameters



Fertility risks use a time partition to define the columns. For the age baseline we use the partition AGEINT_STATE that was defined in PersonCore.mpp. The possible union states for the relative risk factors use the classification UNION_STATE which is declared in PersonCore.mpp as well.

```
parameters
{
	//EN Age baseline for first pregnancy
	double AgeBaselinePreg1[AGEINT_STATE];
	//EN Relative risks of union status on first pregnancy
	double UnionStatusPreg1[UNION_STATE];
};

parameter_group P02_Ferility		//EN Fertility
{
	AgeBaselinePreg1, UnionStatusPreg1
};
```

Actor declarations

The only state of the fertility module is parity_status, which can only have two levels: 'childless' and 'pregnant'. (This is because RiskPaths no longer simulates an actor's fertility events after first conception).

In Fertility.mpp, we only model one event: pregnancy. The corresponding pair of event functions is timeFirstPregEvent and FirstPregEvent.

```
classification PARITY_STATE                //EN Parity status
{
      PS_CHILDLESS,                        //EN Childless
      PS_PREGNANT                          //EN Pregnant
};

actor Person
{
      //EN Parity status derived from the state parity
      PARITY_STATE parity_status = {PS_CHILDLESS};

      //EN First pregnancy event
      event timeFirstPregEvent, FirstPregEvent;
};
```

Event implementation

As is true with all Modgen events, the first pregnancy event is implemented in two parts. The first determines the timing of the event, the second the consequences if the event happens. The timeFirstPregEvent function verifies if the actor is currently at risk and, if so, draws a random duration based on the underlying piecewise proportional constant hazard regression model parameterized by an age baseline and relative risk by union status. Accordingly, the hazard rate is calculated from the two parameters AgeBaselinePreg1 and UnionStatusPreg1. A random duration can be obtained from a uniform distributed random number by the transformation:

randdur = - log(RandUniform(1)) / hazard.

The Modgen function RandUniform() returns a uniform distributed random number between 0-1. The function takes an integer argument used to assign a different independent random number stream to each random number function in the code. When omitted, Modgen automatically writes back a unique index into the .mpp file before translation into C++ code.

When the event happens, the state "parity" is increased by 1. (Note that the derived state parity_status is changed to "PS_PREGNANT" automatically).

```
TIME Person::timeFirstPregEvent()
{
      double dHazard = 0;
      TIME event_time = TIME_INFINITE;
      if (parity_status == PS_CHILDLESS)
```

```
      {

            dHazard = AgeBaselinePreg1[age_status]
              * UnionStatusPreg1[union_status];
          if (dHazard > 0)
          {
                event_time = WAIT(-log(RandUniform(1)) / dHazard);
            }
  }
  return event_time;
}

void Person::FirstPregEvent()
{
      parity_status – PS_PREGNANT;

}
```

7.3.3 Unions.mpp

The programming of union transitions introduces only minor new concepts in Modgen programming--thus, the following code discussion is mainly limited to union dissolutions. The hazard rates for both first and second union dissolution events are stored in the same parameter table, as they each use the same time intervals of union duration.

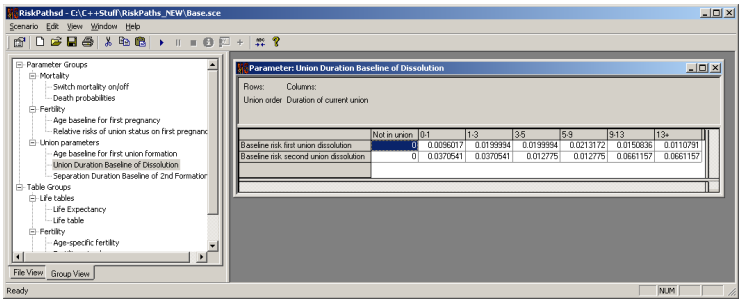
In order to construct a parameter with the dimensions time and union order, we define a time partition and a classification:

```
partition UNION_DURATION            //EN Duration of current union
{
      1, 3, 5, 9, 13
};

classification UNION_ORDER          //EN Union order
{
      UO_FIRST,         //EN First union
      UO_SECOND         //EN Second union
};

parameters
{
      …
      //EN Union Duration Baseline of Dissolution
      double UnionDurationBaseline[UNION_ORDER][UNION_DUR];
      …
};
```

Figure 9: Union dissolution parameters



In the timeUnion1DissolutionEvent() function, hazard rates for first union dissolution are obtained as:

```
dHazard = UnionDurationBaseline[UO_FIRST][union_duration];
```

Accordingly, timeUnion2DissolutionEvent() references the second row from the parameter:

```
dHazard = UnionDurationBaseline[UO_SECOND][union_duration];
```

As opposed to the processes discussed so far, the union dissolution processes do not start at a predefined time (e.g. the 15th birthday) but at union formation events. The union duration spell is defined as a derived self-scheduling state in the following form:

```
//EN Currently in an union
logical in_union = (union_status == US_FIRST_UNION_PERIOD1
      || union_status == US_FIRST_UNION_PERIOD2
      || union_status == US_SECOND_UNION);

//EN Time interval since union formation
int   union_duration = self_scheduling_split(
            active_spell_duration( in_union, TRUE), UNION_DURATION);
```

With respect to union formation, the implementation of the clock which changes the union duration state union_status from US_FIRST_UNION_PERIOD1 to US_FIRST_UNION_PERIOD2 after three years in a first union deserves some discussion. In contrast to the self-scheduling derived states used for all other clocks of the model, here – mainly as an illustration of this alternative - we explicitly implement the clock as an event itself. This event occurs after three years in the first union. The clock is set at first union formation. The actor declaration includes a state which records the time of the status change as well as the event declaration.

```
actor Person
{
      …
      //EN Time of union period change
      TIME  union_period2_change = {TIME_INFINITE};
```

```
      //EN Union period change event
      event timeUnionPeriod2Event, UnionPeriod2Event;
};
```

The time for the state change is set in the first union formation event. In the code sample, WAIT is a built-in Modgen function that returns the time of the current event, plus a specified time (in our example, three years).

```
void Person::Union1FormationEvent()
{
      unions++;
      union_status = US_FIRST_UNION_PERIOD1;
      union_period2_change = WAIT(3);
}
```

The event implementation is straight forward:

```
TIME Person::timeUnionPeriod2Event()
{
      return union_period2_change;
}

void Person::UnionPeriod2Event()
{
      if (union_status == US_FIRST_UNION_PERIOD1)
      {
            union_status = US_FIRST_UNION_PERIOD2;
      }
      union_period2_change = TIME_INFINITE;
}
```

7.4 Tables.mpp

Modgen provides a very powerful and flexible cross-tabulation facility to report model results. The programming of each output table usually requires only a few lines of code. RiskPaths contains only one table file which contains the declarations of all of its output tables—however, for more detailed models, it is advisable to split up table declarations by behavioural groups.

The basic syntax for tables is displayed in Figure 6. The two central elements of a table declaration are the captured classificatory dimensions (defining when an actor enters and leaves a cell) and the analysis dimension (recording what happens while an actor is in that cell). Typical classificatory dimensions are age or time intervals (e.g. fertility by age), states (e.g. fertility by union status), or a combination of both. Modgen does not limit the number of dimensions.

The analysis dimension can contain many expressions, which can be states or derived states. Modgen provides a very useful list of special derived state functions which record, for example, the number of occurrences of certain events, the number of changes in states, or the duration in states. Two particularly helpful concepts are the keyword **unit** and the derived state function

duration() -- unit records the number of actors entering a table cell whereas **duration()** records the total time an actor stayed in the cell.

Tables can contain filter criteria for defining if and under which conditions actor characteristics will be recorded. The Modgen table concepts are best understood by concrete examples as given below. As the full wealth of the Modgen table language goes beyond the scope of this chapter, you are also invited to consult the Modgen Developer's Guide.

Figure 6: Table Syntax

```
table actor_name table_name                          //EN table label
[filter_criteria]
{
      dimension_a *                                  //EN dimension label
      …
      {
            analysis_dimension_expression_x,         //EN expression label
            …
      }
      * dimension_n                                  //EN dimension label
      …
};
```

7.4.1 Table 1: Life expectancy

The first table example contains summary values of our simulation and has no dimensions, i.e. cells apply to the entire population over the entire simulation period. We make use of the Modgen keyword **unit**, which counts the number of actors entering the cell of a table (in our example, the simulation itself), and the Modgen function **duration()** which sums up the time actors stay in this cell (in our example, the total years lived by all actors in the simulation). The average age at death of all actors in the simulation is then obtained by dividing **duration()** by **unit**. As for parameter declarations, comments placed in the code are used as labels in the application. (Note that in the table declaration below, the 'decimals=3' portion of the comment is used to determine the number of decimal places in the table; this part of the comment does not carry through to the label used in the report).

```
table Person T01_LifeExpectancy //EN 1) Life Expectancy
{
 {
      unit,                    // EN Total simulated cases
      duration(),              // EN Total duration
      duration()/unit          // EN Life expectancy decimals=3
 }
};
```

7.4.2 Table 2: Life table

In the second table we record the population by age. For output by age, we use integer_age as table dimension.

```
table Person T02_TotalPopulationByYear    //EN Life table
{
      //EN Age
      integer_age *
      {
            unit,                         //EN Population start of year
            duration()                    //EN Average population in year
      }
};
```

Unit and **duration()** now refer to the number of entrances into - and durations within - one year age intervals. **Unit** thus counts the actors present at the beginning of each year, while **duration()** refers to the average population in the year.

7.4.3 Tables 3 and 4: Age-specific fertility

As well as the keywords **unit** and the derived state function **duration()**, states and a set of other derived state functions can be used in tables. If using a state without a function, Modgen records the change of the state while in a particular cell, i.e. the value of the state when the cell is exited minus the value of the state when the cell was entered.

The expression transitions(parity_status, PS_CHILDLESS, PS_PREGNANT) / duration() records the (age specific) fertility as the number of birth events divided by the average number of women by year of age.

The second expression is used to calculate the true rate, i.e. the number of birth events by exposure time. A woman is under exposure for first pregnancy when childless. We thus divide the number of events by the term ‘duration(parity_status, PS_CHILDLESS)’.

The table dimension is age in full years. As fertility is 0 until age 15 and very low after 40, the age periods before 15 and after 40 are not further divided. We thus define a partition AGE_FERTILEYEARS which is used in the self_scheduling_split which defines the table dimension.

```
partition AGE_FERTILEYEARS //EN Fertile age partition
{
      15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
      32, 33, 34, 35, 36, 37, 38, 39, 40
};

table Person T03_FertilityByAge     //EN Age-specific fertility
{
      //EN Age
      self_scheduling_split(age,AGE_FERTILEYEARS) *
```

```
      {
            //EN First birth rate all women decimals=4
            transitions( parity_status, PS_CHILDLESS, PS_PREGNANT ) /
                  duration() ,

            //EN First birth rate woman at risk decimals=4
            transitions( parity_status, PS_CHILDLESS, PS_PREGNANT ) /
                   duration( parity_status, PS_CHILDLESS )
      }
};
```

Table 4 produces first birth rates by the 2.5 year age groups used for parameterization. We also add an additional dimension, namely the union status; we thus obtain simulated values of the model parameters.

```
table Person T04_FertilityRatesByAgeGroup //EN Fertility rates by age group
[parity_status == PS_CHILDLESS]
{
      {
            //EN Fertility decimals=4
            transitions( parity_status, PS_CHILDLESS, PS_PREGNANT ) /
                  duration()
      }
      * self_scheduling_split(age, AGEINT_STATE)      //EN Age interval
      * union_status                                  //EN Union Status
};
```

7.4.4 Table 5: Cohort fertility

Table 5 calculates two cohort measures of fertility -- average age at first conception and childlessness. To obtain the age at pregnancy we use the Modgen derived state function value_at_transitions(parity_status, PS_CHILDLESS ,PS_PREGNANT, age) which returns the value of one state (age) at a specific transition of another state, namely when parity_status changes from PS_CHILDLESS to PS_PREGNANT.

```
table Person T05_CohortFertility          //EN Cohort fertility
{
      {
      //EN Av. age at 1st pregnancy decimals=2
      value_at_transitions(parity_status,PS_CHILDLESS,PS_PREGNANT,age)/
            transitions( parity_status, PS_CHILDLESS, PS_PREGNANT ),

      //EN Childlessness decimals=4
      1 - transitions( parity_status, PS_CHILDLESS, PS_PREGNANT ) / unit,

      //EN Percent one child decimals=4
            transitions( parity_status, PS_CHILDLESS, PS_PREGNANT ) / unit
      }
};
```

7.4.5 Table 6: Pregnancies by union status and order

In table 6 we use an example of a filter which triggers a person exactly at the entrance of a state, in our case at the occurrence of pregnancy. We are interested in the union status at first conception. Note that this filter also excludes women who stay childless.

```
table Person T06_BirthsByUnion //EN Pregnancies by union status & order
[trigger_entrances(parity_status, PS_PREGNANT)]
{
      {
            unit        //EN Number of pregnancies
      }
      *union_status+    //EN Union Status at pregnancy
};
```

7.4.6 Table 7: First union formation risks

Like table 4 this table reproduces a parameter table. While such an output table does not contain any information (for a sufficiently large sample size it will come close to the original model parameters) it is useful for model validation and to assess Monte Carlo variability.

```
table Person T07_FirstUnionFormation      //EN First union formation
[parity_status == PS_CHILDLESS]
{
      //EN Age group
      self_scheduling_split(age, AGEINT_STATE) *
      {
            //EN First union formation risk decimals=4
            entrances(union_status, US_FIRST_UNION_PERIOD1)
            / duration(union_status, US_NEVER_IN_UNION)
      }
};
```

7.4.7 Grouping of table output

Like parameters, output tables can also be grouped for a more meaningful presentation of results. In the application RiskPaths, we distinguish three groups of tables: life tables, fertility tables, and tables for union status.

```
table_group TG01_Life_Tables        //EN Life tables
{
 T01_LifeExpectancy, T02_TotalPopulationByYear
};

table_group TG02_Birth_Tables       //EN Fertility
{
      T03_FertilityByAge, T04_FertilityRatesByAgeGroup, T05_CohortFertility
};

table_group TG03_Union_Tables       //EN Unions
{
      T06_BirthsByUnion, T07_FirstUnionFormation
```

```
};
```

7.5 Tracking.mpp

The track{} code block defines the list of states to be recorded longitudinally for visual BioBrowser output. This command is frequently placed in table files. In our model, however, we have decided to code a separate Tracking.mpp file, since we also track risk patterns calculated as derived states.

```
track Person
{
      integer_age,
      life_status,
      age_status,
      union_duration,
      dissolution_duration,
      unions,
      parity_status,
      union_status,
      preg_hazard,
      formation_hazard,
      dissolution_hazard
};
```

The file also includes the declaration of three derived states. We have used the derived state concept to calculate the three main hazard rates (pregnancy, union formation, and union dissolution) for BioBrowser output. They are for illustrative purposes only, as all hazard rates, broken down by union order, are calculated in the event functions.

The declaration of the derived states preg_hazard, formation_hazard, and dissolution_hazard are also good syntax examples of how derived states can be built from simple states by if-else constructs.

```
actor Person
{
      //EN Pregnancy hazard
      double preg_hazard = (parity_status == PS_CHILDLESS) ?
            AgeBaselinePreg1[age_status] *
            UnionStatusPreg1[union_status] : 0;

      //EN Union formation hazard
      double formation_hazard = (union_status != US_NEVER_IN_UNION
            && union_status != US_AFTER_FIRST_UNION) ? 0 :
            ((union_status == US_NEVER_IN_UNION) ?
            AgeBaselineForm1[age_status] :
            SeparationDurationBaseline[dissolution_duration] );

      //EN Union dissolution hazard
      double dissolution_hazard = (union_status != US_FIRST_UNION_PERIOD1 &&
```

```
            union_status != US_FIRST_UNION_PERIOD2 &&
            union_status != US_SECOND_UNION) ? 0 :
            ((union_status == US_SECOND_UNION) ?
            UnionDurationBaseline[UO_SECOND][union_duration] :
            UnionDurationBaseline[UO_FIRST][union_duration]);
};
```

7.6 Language translation file RiskPathsFR.mpp

This .mpp file will only exist for models that are defined in Modgen to be multilingual (which for RiskPaths implies English and French). Even for a bilingual model, however, one of English or French is still deemed to be the first or primary language of the model. English was chosen as the primary language when RiskPaths was originally developed, and so the RiskPathsFR.mpp file essentially contains translations for the model's labels and notes in the other language, i.e. French. (If the original primary language of RiskPaths had been French, this translation file would have been called RiskPathsEN.mpp and it would have contained English translations of the labels and notes for the model.)

Normally, all notes and labels are entered as code comments in the source .mpp files, using the primary language of the model, as has been illustrated several times in the previous examples. The corresponding translations are subsequently placed in this separate .mpp file.

References

Thomson , E., M. Winkler-Dworak , M. Spielauer , A. Prskawetz (2009) Union Instability as an Engine of Fertility? A Micro-simulation Model for France. Vienna Institute for Demography working paper 2009-02 http://www.oeaw.ac.at/vid/download/WP2009_02.pdf

Alain Bélanger, A, Jean Dominique Morency, M. Spielauer (2009). The Impact of Union Stability on Cohort Fertility Variations in Canada: A Microsimulation Approach. Paper presented at Canadian Population Society's Conference, Ottawa, 2009.