

Dynamic Micro-Simulation for Population Projection - An Application to Mauritania

Release 1.0

Martin Spielauer and Olivier Dupriez

June 16, 2017

Table of Contents

Dynamic Micro-Simulation for Population Projections - An Application to Mauritania	3
Acknowledgments	5
Abstract	7
Chapter 1. Rationale, Methods, Implementation, and a Portable Application	9
1.1. Introduction: Purpose, Scope, Organization	9
1.2. The Dynamic Micro-simulation Approach	11
1.3. Population Projections by Micro-simulation: Rationale	16
1.4. A Portable Modular Application	17
1.5. Data Requirements	26
Chapter 2. Phase 1: Reproducing a Typical Macro Population Projection Model	31
2.1. The Base model: Reproducing a macro model	31
2.2. Starting Population	32
2.3. Fertility	34
2.4. Mortality	35
2.5. Internal Migration	37
2.6. Immigration	39
2.7. Emigration	42
Chapter 3. Phase 2: Extending the Model Showcasing the Power of Micro-simulation	45
3.1. Primary Education	46
3.2. Union Formation	51
3.3. Alternative Models for Fertility	56
3.4. Infant Mortality	59
3.5. Migration refinements	63
Chapter 4. Using the model	65
4.1. Analysis Example: Demographic Effects of Education	65
4.2. Analysis Example: De-Composing Changes in Child Mortality	70
4.3. Analysis Example: Internal Migration by Education	72
Chapter 5. Micro-simulation Technology: Modgen/OpenM++	77
5.1. The User's Perspective: User Interface	78

5.2. The Developer’s Perspective: Key Programming Concepts	86
Chapter 6. Statistical Measures, Concepts, and Methods Common in Micro-simulation	91
6.1. Starting a Simulation and Keeping it Going: Population, States, Events	91
6.2. Measures of the likelihood of events: probabilities and rates	92
6.3. From Measures to a Simulation	93
6.4. Estimating and calibrating probabilities and rates	94
6.5. Probability Distributions and Origin-destination Matrices	96
6.6. Micro-simulation Implementation	96
Chapter 7. A Step-by-Step Guide	99
7.1. Step 1: Model Templates	100
7.2. Step 2: Creating a Population	109
7.3. Step 3: Population Scaling	117
7.4. Step 4: Fertility	122
7.5. Step 5: Refining Mortality	128
7.6. Step 6: Migration	132
7.7. Step 7: Emigration	137
7.8. Step 8: Immigration	140
7.9. Step 9: Micro-Data Output	151
7.10. Step 10: Micro-Data Input	153
7.11. Step 11: Conversion to a Time-Based Model	156
7.12. Step 12: Sampling from a detailed starting population	162
7.13. Step 13: Primary Education	167
7.14. Step 14: First Union Formation	182
7.15. Step 15: Fertility refined	188
7.16. Step 16: Infant mortality	197
7.17. Step 17: Migration refined	207
7.18. Step 18: Extending Table and Micro-Data Output	208
Chapter 8. Software Downloads	223
8.1. Model Users	223
8.2. Model Developers	223

The DYNAMIS-POP-MRT_v01_2017 model

Dynamic Micro-Simulation for Population Projections - An Application to Mauritania

The model DYNAMIS-POP-MRT-2017_v01



This report and accompanying resources have been developed to demystify micro-simulation for population projections and demonstrate its feasibility and relevance in the context of developing countries. Advanced and freely available programming tools and improvements in the availability and quality of micro-data have helped make dynamic micro-simulation feasible at a reasonable cost. At the core of this report is an application example for Mauritania, which introduces statistical methods used in micro-simulation and micro-simulation programming techniques, and can be adapted and replicated for other countries. The model has an intuitive graphical user interface and runs on a standard personal computer. Its code and statistical analysis files are available to model builders, who can use them as a textbook and toolbox for micro-simulation model development and implementation.

Photo by Michał Huniewicz (CC BY 2.0).

Acknowledgments

This project, led by the World Bank Development Data Group, was funded by the World Bank Knowledge for Change Program (KCP III, grant number TF0A1095) and by the International Household Survey Network Trust Fund from the United Kingdom Department for International Development (DFID, grant number TF011722 executed by the World Bank Development Data Group).

The project aimed to demonstrate how new approaches—synthetic data generation and dynamic micro-simulation—can take advantage and integrate data from multiple sources—in this case population censuses and household surveys—to inform development policies and research on development issues.

This report describes a micro-simulation population projection model applied to Mauritania. The model will be expanded and adapted to other countries. A specialized education module, more advanced than the education module currently available in the model, is being developed and will be integrated in the model. Solutions to exploit geo-spatial data will be developed and integrated in the model, allowing fine geographic disaggregation of the model's input and output. And other specialized modules may be developed. This first application to Mauritania is thus the first of what we envision to become a collection of models to be developed and published as public goods under the generic name DYNAMIS (Dynamic Micro-Simulation).

The project was initiated and led by Olivier Dupriez, Lead Statistician at the World Bank Development Data Group. Martin Spielauer, expert in micro-simulation and consultant for the World Bank, is the main author of the model. Alexander Kowarik, statistician and World Bank consultant, contributed to the project as an expert in synthetic data generation. Staff from the *Centre Mauritanien d'Analyse de Politiques* (CMAP) and from the *Office National de la Statistique de Mauritanie* (ONS) participated in the country adaptation of the model, in the provision and analysis of input data, and in the analysis of the model output. The national counterparts in this research project included El Hassen Zein (Coordinator, CMAP), Cheikh Baye Beddy (Deputy Coordinator, CMAP), Mahjouba Habib (Junior expert, CMAP), Meimouna El Moustapha (Econometrician, CMAP), Yahya Menaya (Econometrician, CMAP), Oumelkhairat Abba (Computer expert, CMAP), Mohamed El Moctar Ahmed Sidi (General Manager, ONS), Taleb Mahjoub (Deputy General Manager, ONS), El Yass Didi (Director, Social and Demographic Statistics, ONS), Mariem Mohamed Saleh (Director, Dissemination and data processing, ONS), Boubekrine Mohamed Kabir (Statistician-demographer, ONS), and Hachim Mohamed Maaloum (Data processing specialist, ONS). Mehmood Asghar (World Bank) provided IT support. The report was copy-edited by Linda Klinger.

The model was developed using MODGEN (a freeware developed and published by Statistics Canada) and Microsoft Visual Studio. Data analysis for the calculation of the model parameters was done using Stata, and a synthetic population dataset (not used in the final version of the model) was produced using the simPop R package maintained by Matthias Templ. The html documentation of the model was built with Sphinx using a theme provided by Read the Docs.

Abstract

Population projections are key for policy making and planning. Currently, most countries, and international organizations like the World Bank and the United Nations, produce population projections using the cohort-component method. While simple and applicable in absence of detailed data sources, the method is limited to projecting a population by age, sex, and very few additional variables, such as province or broad education categories. A more advanced—but still uncommon—approach involves dynamic micro-simulation models, where populations are represented by large samples of individual people and their life-courses over time. This approach is more complex, but has major advantages: it can produce detailed projections of a broad variety of individual characteristics, model realistic life-courses and their diversity, and support the modeling of interactions between people. Such models also support more detailed planning and policy development, and can provide the demographic core of more extensive socioeconomic models. While currently applied almost exclusively in the developed world, the benefits are also evident in the context of developing countries.

This report helps demystify micro-simulation for population projections and demonstrate its feasibility in the developing world. Dynamic micro-simulation is now feasible at a reasonable cost due to advanced and freely available programming tools, as well as improvements in the availability, quality, and standardization of micro-level data. At the core of the report is an application example for Mauritania, which introduces statistical methods used in micro-simulation and micro-simulation programming techniques, and can be adapted and replicated for other countries. The model has an intuitive graphical user interface and runs on a standard PC, and the report includes step-by step documentation of its computer implementation. Its code and statistical analysis files are available to model builders, who can use them as a textbook and toolbox for micro-simulation model development and implementation.

Keywords: Micro-simulation, Population Projection, Development, Mauritania, Dynamis

Chapter 1. Rationale, Methods, Implementation, and a Portable Application

1.1. Introduction: Purpose, Scope, Organization

Population projections are key for policy making and planning. Changes in the size and composition of the population are key determinants of the demand for goods and services, from basic food and education, to energy and housing. Population projections, based on multiple scenarios, help government and other decision makers make informed decisions.

Most countries and international organizations, like the World Bank and the United Nations, produce population projections using the cohort-component method, a macro approach limited to a very small number of characteristics. This provides projections by age and sex at the national level, often disaggregated by urban/rural areas; large countries sometimes provide projections at a sub-national level. Given the high importance of education on human capital and its influence as the “single most important variable besides age and sex” on demographic behaviors (Lutz et.al. 1999), population projections that included education later became available for most countries, but that extension defined the technical limit of this approach.

A more advanced—but still uncommon—approach consists of dynamic micro-simulation models, in which populations are represented by large samples of individual people and their life-courses over time. This approach is more complex, but it has major advantages: it can produce detailed projections of a broad variety of individual characteristics, model realistic life-courses and their diversity, and support the modeling of interactions between people. The idea is to start with a micro-data population and make it evolve over time, unlike the cohort-component approach, which starts with a simple distribution of a population by age and sex but cannot track individuals over time. The idea is not new (van Imhoff and Post, 1998), but it has become feasible only recently, because advances in freely available programming technologies and improvements in available data have reduced costs. Micro-simulation models also allow more disaggregation and thus can provide projections of the population or some of its components, e.g., population by ethnic affiliation, school-age population by region, labor force by education level, etc. In addition, population micro-simulation models can provide the demographic core and foundation for more specialized micro-simulation models of diseases, tax benefit models, pension models, etc. A micro-simulation population projection model is used by Statistics Canada to project the diversity of the Canadian population by visible minority group (Caron-Malenfant et.al. 2010), Aboriginal identity (Morency et.al. 2015), and labor force (Martel et.al. 2011), as well as to study the effect of educational improvements on the future size and composition of the Aboriginal labor force (Spielauer 2014).

Population micro-simulation can be seen both as a complement to macro-projections and as their replacement, as macro-projection models can be implemented as micro-simulations, producing identical results, but allowing for model extensions beyond the technical limitations of cohort-component models. This point is demonstrated in this report with a micro-simulation example developed in two phases. The first

phase reproduces the macro model DemProj (Stover & Kirmeyer 2001), which is widely used, especially in developing countries. In our example, we use Mauritanian data for its parameterization. In the second phase, the model is extended to incorporate more detailed characteristics and behaviors. Again we based the model on Mauritanian data sources, for which equivalent data are available throughout the world. Extensions include the fertility module, which allows a realistic projection of family sizes and fertility by education. We also added a module for child mortality, incorporating factors such as the mother's education and number of siblings. Education is modeled to include its inter-generational transmission; first union formation is introduced as key determinant of the timing of first births.

This report is an attempt to demystify micro-simulation and, to a lesser extent, synthetic population, and demonstrate its feasibility in developing countries. The Mauritania example is key to this report, and includes detailed instructions for its adaptation and replication in other countries. The model is implemented in Modgen, a freely available software package developed and maintained at Statistics Canada. This package is compatible with a recent open source implementation of the same programming language under the name openM++. For the baseline population, we used a synthetic (or "virtual") population generated using simPop (an open source R package). This approach allowed us to combine information from various data sources (in our case, the 2013 Population and Housing Census and the Multiple Indicators Cluster Survey 2011) into one data file and make these micro-data non-confidential.

The model also introduces micro-simulation programming and various statistical methods used in micro-simulation, and includes step-by-step documentation of its computer implementation. The model has an intuitive graphical user interface and runs on a standard PC. Its code and statistical analysis files are openly accessible and can be used for micro-simulation model development and implementation. New software solutions, which are free, and improved computer capabilities make it an affordable and much more feasible option for developing countries. Given the growing interest in disaggregated data for development planning and monitoring (for example to monitor the Sustainable Development Goals), it makes sense to also push for more disaggregation in projections and simulations. Besides producing more detailed projections, micro-simulation also allows a more explicit incorporation of theory and policy levers in its projections. For example, it allows modeling of inter-generational dynamics and analysis of downstream effects on demographic change or child mortality of policy interventions that improve education.

This report has four parts:

- **Background and Overview:** The first part, *Chapter 1. Rationale, Methods, Implementation and a Portable Application*, introduces dynamic micro-simulation and its rationale for population projections, defines data requirements, and introduces the portable application and its user interface.
- **Application Example:** The second part details the portable model example, and *Chapter 2. Reproducing a Typical Macro Population Projection Model* and *Chapter 3. Extending the Model Showcasing the Power of Microsimulation* describe the population projection model applied to Mauritania. The first phase reproduces a typical macro model. In a second phase, we add modules, variables, and processes beyond the macro framework to illustrate key features and strengths of the micro-simulation approach. Parameter tables and the required data analysis for each module that replicates the calculation of model parameters are discussed. Links to all Stata files and a set of Excel files are also provided. The result is both a fully functional model that can be customized for other countries, as well as an illustration of typical modeling approaches found in micro-simulation. The model can also be understood as a modeling platform, allowing for extensions. Some modules contain parallel implementations of the user's modeling options, which allows for a wide range of scenarios and assumptions. Scenario building and use of the model for projection analysis is demonstrated in *Chapter 4. Using the model*.
- **Technology:** This part discusses technological aspects of building micro-simulation models. *Chapter 5. Microsimulation Technology: Modgen/OpenM++* discusses the technical implementation of the model from both a user's and a developer's perspective, thereby introducing the key components of the Modgen/OpenM++ programming technology and resulting applications. *Chapter 6. Statistical Models Common in Microsimulation* discusses the statistical models and concepts used in the example and

introduces key methods widely used in micro-simulation.

- **Materials:** This part provides all materials to reproduce the sample model. *Chapter 7. A Step-by-Step Guide for Implementing DYNAMIS-POP-MRT* is a detailed documentation of the model code that model builders can use for micro-simulation model development and implementation. Each step discusses the model functionality, explains new programming concepts, and discusses the programming code. Finally, *Chapter 8. Software Downloads* describes and provides all necessary software downloads, including the micro-simulation model, programming resources, and statistical analysis files.

Data requirements are met by many countries—through their population censuses, and implementation of demographic household surveys like UNICEF’s Multiple Indicators Cluster Surveys (MICS) and Demographic and Health Surveys (DHS)—and we expect improvements on specialized modules. Once built and compiled, using the model is easy, as it has an intuitive graphical user interface and runs on a standard PC. While model development remains a complex issue, a collection of modular components that can be adapted and assembled for various purposes is being built. While it will not be a plug-and-play tool and still require technical expertise for model development or customization, this collection of well-documented modules should save developers and data analysts considerable time and lower the entry barrier into micro-simulation modeling.

References

Caron-Malenfant, E.; Lebel, A. & Martel, L. (2010), ***Projections of the Diversity of the Canadian Population 2006 to 2031***, Statistics Canada Catalogue no. 91-551-X. [PDF]

Lutz, W., J.W. Vaupel, and D.A. Ahlburg, Eds. (1999) ***Frontiers of Population Forecasting***. A Supplement to Vol. 24, 1998, Population and Development Review. New York: The Population Council.

Martel, L., É. Caron Malenfant, J.-D. Morency, A. Lebel, A. Bélanger, N. Bastien (2011) ***Projected trends to 2031 for the Canadian labour force***, Statistics Canada Catalogue no. 11-010-X, vol. 24, no. 8 [HTML]

Morency, J-D, É. Caron-Malenfant, S. Coulombe, S. Langlois (2015) ***Projections of the Aboriginal Population and Households in Canada, 2011 to 2036*** - Statistics Canada Catalogue no. 91-552-X [PDF]

Spielauer, M. (2014), ***The relation between education and labour force participation of Aboriginal peoples: A simulation analysis using the Demosim population projection model***, Canadian Studies in Population 41(1-2), 144–164.[PDF]

Stover, J., S. Kirmeyer (2001), ***DemProj Version 4 - A Computer Program for Making Population Projections***, The POLICY Project Spectrum [PDF]

Van Imhoff, E. & W. Post (1998), ***Microsimulation methods for population projection***, Population 10(1), 97–136. [PDF]

1.2. The Dynamic Micro-simulation Approach

Micro-simulation in the context of socioeconomic applications can be perceived as an experiment with a virtual society of thousands—or millions—of individuals. Micro-simulation models can be static or dynamic. Central to dynamic micro-simulation is the explicit modeling of the time dimension, following people and their families or households over time, rather than performing a before-and-after comparison in response to changes in the tax-benefit system or economic shocks. Dynamic modeling lends itself naturally to the modeling of policies with a longitudinal component, e.g., educational investments, especially in the context of general rapid social, economic, and demographic change that make it difficult to assess the contribution of individual policies to overall trends without tracking and comparing the lives of individuals who form a society.

This report advocates for dynamic micro-simulation, which has become increasingly feasible due to tech-

nical advances. Dynamic micro-simulation complements traditional evaluations of effects of development programs, given the context of the many development issues that evolve and impact individuals, families, and households over time and generations.

1.2.1. Advantages

Micro-simulation is attractive both from a theoretical and a practical point of view, as it supports research embedded into modern paradigms, such as the life-course perspective, while simultaneously providing a tool for what-if analysis of high policy relevance. Typical application areas include tax-benefit analysis, analysis of pension system adequacy and sustainability, and health and health insurance. One recent development is using micro-simulation in demographic projections. Although this has been discussed in literature for almost two decades (e.g., Imhoff & Post 1998), larger-scale implementations are a recent development. Statistics Canada was the first statistical office to produce official population projections using micro-simulation. Called Demosim, this model is implemented using the micro-simulation programming technology Modgen, which Statistics Canada developed. Modgen is freely available and shared worldwide. Variants of Demosim are currently being developed for several European countries as well as Australia (Belanger, forthcoming).

In social sciences and economics, there is an increasing emphasis on processes rather than static structures. This is where dynamic micro-simulation is most effective, as it can simultaneously deal with distributional and dynamic issues, e.g., demographic change, and with a longitudinal dimension to distributional analysis. From a policy perspective, there is increasing emphasis on processes rather than static structures; it also reflects the modern way of assessing poverty as a multi-dimensional and dynamic phenomena to be addressed by policies that go beyond static redistribution of resources, enable people to leave this state, and reduce poverty risks permanently. Micro-simulation can improve understanding of the complex dynamics resulting from many simultaneous processes, which are often studied only in isolation. In this context, dynamic micro-simulation is a key component in the evolution from analysis to synthesis (Willekens 2001) and can integrate analyses of single processes into computer simulations of societies. Such models can then be used for what-if analysis and assess how the future is shaped by today's decisions, policies, and actions.

In principle, whenever a system is made up of small-scale units, micro-simulation is an optional simulation approach. Micro-simulation has the potential to be especially powerful in three scenarios (Spielauer 2011): population heterogeneity, aggregation of behavioral relations, and individual histories.

Population heterogeneity

Micro-simulation is the preferred modeling choice when population heterogeneity matters and there are too many possible combinations of characteristics to split the population into a manageable number of groups.

Using a large sample of members to represent a population is an intuitive way to capture the population's diversity and allows for distributional analysis of the effects of changes and reforms. Creating such a representation in a database of individuals is a typical step in micro-simulation development. Frequently, it will combine and integrate data from various sources, which ultimately makes data more relevant for policy analysis. It also allows policies to be targeted to very specific segments of the population.

From a longitudinal perspective, micro-simulation can capture the variety and heterogeneity in life-course experiences and careers, adding a whole new dimension in distributional analysis with the use of lifetime measures and capture of distributional impacts of policies over the individual life-cycle and between cohorts and generations. Besides its typical use for distributional analysis, micro-simulation can also track different ethnicities or minorities, which often display persistent behavioral differences over time, such as demographic behaviors. Individual modeling can improve the accuracy of projections. Besides persistence in differences, another frequently observed or theorized phenomenon (e.g., in modernization theory) is that behavioral changes are adapted by different groups at different times, and that certain sectors of society lead this development and are followed by others. The ability to model such processes can improve the theoretical foundation of projections.

The problem of aggregation

Micro-simulation is an adequate modeling choice if behaviors are complex at the macro level but better understood at the micro level.

Unlike macro models, micro-simulation does not require behaviors to be aggregated, but aggregates the outcomes of individual behaviors. It is not bound by restrictive assumptions necessary for representing society by a representative agent or small group of agents. From a static accounting perspective, tax and social security regulations tie rules in a non-linear way to individual and family characteristics, impeding the aggregation of their operations. To calculate total tax revenues or costs of means-tested policies, we need to know composition of the population by income (progressive taxes), family characteristics (dependent children and spouses) and all other characteristics that affect the calculation of individual liability or eligibility.

In dynamic systems, many behaviors are modeled much more easily at the micro level, as this is where decisions are made. In many cases, behaviors are also more stable at the micro level, where there is no interference from composition effects. Even complete stability at the micro level does not automatically correspond to stability at the macro level. For example, educational attainments of the current school age population might be stable for given geographical, ethnic, and parental characteristics, but the composition of the population changes over time and may be further affected by migration and other population changes.

Based on (and producing) micro-data, micro-simulation allows flexible aggregation, as the information may be cross-tabulated in any form. As aggregation schemes do not have to be determined a priori, micro-simulation can develop and apply a broad range of output measures. This directly benefits the measurement of complex issues like income adequacy at old age, poverty as a very multidimensional phenomenon, or a wide variety of measures developed in literature.

Individual and linked histories

Dynamic micro-simulation is the only modeling choice if individual histories matter, i.e., when processes have memory.

Individual histories can become important factors in policy analysis, as they influence behaviors (e.g., a cash transfer may enable families to send their children to school, which generates an education history impacting their lives in many dimensions); affect risks (e.g., mortality by smoking histories); and are the foundations of many accounting issues (e.g., pensions that depend on individual contribution histories). Keeping memory allows measures like durations in states (e.g., healthy life, time worked, time spent in care institutions) and tallies of experiences (e.g. visits to hospitals, death of a child).

Keeping individual histories of income, taxes, and benefits can be useful, specifically for cost-benefit analysis, as they can distinguish between private and social return on investments such as education. As micro-simulation can link actors to families or households, it can extend histories over generations and help to better assess policies with long-term downstream effects. For example, enabling a person to attain higher education will not only lead to higher individual wages, it could affect expected tax payments and benefits received over the life-course, as well as family formation, number of children, and child mortality, education, and poverty risks, potentially stopping the inter-generational transmission of poverty.

1.2.2. Drawbacks

Limitations and drawbacks of micro-simulation can be classified in two categories: those that are intrinsic to all modeling and efforts to make statements about the future, especially the trade-off between detail and prediction power; and those that are transitory, as they can be expected to keep decreasing over time, such as costs of hardware, technical requirements, and data availability and quality issues.

Detail versus prediction power

The central limitation of micro-simulation is that the degree of model detail does not go hand-in-hand

with overall prediction power. Providing more detailed models, something at which micro-simulation excels, does not necessarily mean the models are “better.” The ability to produce distributions comes at the price of losing predictive power in projecting means, and the ability to make very accurate statements in the short run does not necessarily lead to models that are useful for long-term projections. An analogy is weather forecasts: detailed models for the weather tomorrow, on a geographical scale, will not be of use for the projection of global climate changes over the next centuries. This also applies to socioeconomic models. The longer the time horizon and the more important the mean, the more the focus should be directed to the main driving forces and a solid theoretical foundation of these mechanisms. The reason for this can be found in what is called randomness, caused by accumulated errors and biases of variable values (for a discussion of randomness in micro-simulation, see Imhoff & Post 1998). In static models, this primarily involves the population database, which typically has to be constructed by combining information from various, and not necessarily consistent, data sources. In dynamic models, randomness is further increased by the stochastic nature of micro-simulation models and the fact that all right-hand variables used in equations for future behaviors have to be simulated as well.

The randomness resulting from the stochastic nature of dynamic micro-simulation is called Monte Carlo variability. Micro-simulation produces not expected values, but random variables distributed around the expected values. Every simulation experiment will produce different aggregate results. While this was cumbersome in the past, when computer capabilities were limited, many repeat experiments and/or the simulation of large populations can reduce this randomness and deliver valuable information on the distribution of results and point estimates.

A more fundamental problem lies in the trade-off between the additional randomness introduced by additional variables and misspecification errors caused by models that are too simplified. This means that the large number of variables that models can include, which is the feature that makes micro-simulation especially attractive, comes at the price of randomness and a decrease in prediction power that occurs as the number of variables increases. Modelers should be aware that this generates a trade-off between good aggregate predictions versus a good prediction regarding distributional issues in the long run. This trade-off is not specific to micro-simulation, but as micro-simulation is frequently employed for detailed projections, the scope for randomness becomes accordingly large.

There are basically two ways of dealing with this trade-off. The first is to keep models simple. The second is to combine the strengths of different modeling approaches. Not surprisingly, in many large-scale micro-simulation models, some outcomes are aligned or calibrated towards aggregated numbers or projections obtained by external means. For example, micro-simulation models may be powerful in modeling the effects of unemployment on individual lives, but will typically use aggregate unemployment rates stemming from other projections or scenarios. Technically, micro-simulation models can be separate, using results from macro models or projection scenarios as input parameters, or they can be linked to macro models, allowing feedback in both directions. Literature is particularly full of examples of linking static micro-simulation models with computable general equilibrium (CGE) models.

The effort to keep models simple often leads to macro models bypassing micro-simulation as a modeling strategy, the choice often justified with the higher development costs of micro-simulation. This choice ignores the fact that micro-simulation can often reproduce the results of macro models if needed (and at comparable costs), while also allowing for step-wise refinements and removal of simplifying assumptions inherent to macro models. This is best illustrated by population projection models, which are to date almost exclusively based on the cohort-component method limited to very few variables—a number probably too small to be justified from a theoretical point of view.

Transitory limitations

An often-stated drawback of micro-simulation is that such models have high data demands and costs typically involve acquiring and compiling such data. It can be noted, however, that such costs are not explicit costs associated with the micro-simulation itself, but represent the price to be paid for research in general, and informed policy making in particular. Recent advances in data availability in its various forms, from administrative data sets being made more accessible for researching internationally standardized survey data could turn this argument around: micro-simulation can make available data more policy

relevant, as it complements traditional data analysis and combines such analysis with a what-if projection tool. In the case of population projections, required data are readily available for many countries and the model can be very generic as the input and output (i.e., requirements and purpose) of the model are very much the same across countries.

Historically, micro-simulation models require large investments with respect to both manpower and hardware. These costs can be expected to decrease over time, however, as hardware prices fall and more powerful and efficient computer languages become available. Development costs can be dramatically cut as technologies become available that do not require that models be built and programmed from scratch. Dramatic efficiency gains have been demonstrated by advances in programming technologies. Modgen, which was used for the application developed in this study, has made model implementation a straightforward process for the next generation of social scientists in the same way that the use of statistical software has put a series of statistical analysis into the toolbox of social scientists.

1.2.3. Types of dynamic micro-simulation models

Micro-simulation models come in many types and flavors. In scope and complexity, they range from models that address specific research questions to multi-purpose models covering a multitude of life-course domains, e.g., education, work, family life, income, saving, retirement, health, and eventually death, together with detailed accounting routines depicting tax-benefit systems and social insurance. While some research questions require complex models—e.g., pension analysis, which requires knowledge of detailed individual life-courses—other applications specialize in specific behaviors. All models have a demographic core, which itself can be designed as a specialized application for population projections as well as a foundation for applications added step-wise in a modular way. The latter is the development strategy used here.

A second distinction concerns the timeframe: dynamic models can operate in discrete time, like years, or in continuous time, allowing events to happen at any moment of time. Discrete time models are the more conventional approach, but it comes with serious drawbacks: when updating states on a yearly basis, information on when, in which order, and how often events happened gets lost. For example, a person may have experienced various episodes of unemployment during a year, but may be employed at the captured time points. We chose a continuous timeframe for our application, which can be implemented very efficiently using Modgen. It is the more flexible approach; developers choose how to model behaviors and when to update states. While some behaviors will be modeled in continuous time, other updates and calculations can still be made in yearly steps.

A third distinction concerns the model's execution: one person or group of persons (e.g., families) at a time, or the whole population at once. The first approach is called case-based. It allows easy parallelization of the model's execution and simulation of huge populations, because the whole population does not have to be kept in memory at each moment of time, as it does in time-based models. In contrast, time-based models allow modeling interactions between all actors and not just within a case. For example, persons can search for spouses within the population. Time-based models also allow aggregation on the fly, which is useful for policies that depend on outcomes (e.g., adjustment of tax rates for balancing books) or if one wishes to align aggregated outputs to given targets (e.g., adjustment of fertility risks to produce a target number of births). Modgen supports both approaches and, when starting as a case-based approach, can easily switch to a time-based approach. This is demonstrated in the application development in this report.

References

- Spielauer, M. (2011), ***What is social science microsimulation?***, Social Science Computer Review 29(1), 9–20. [PDF]
- Van Imhoff, E. & Post, W. (1998), ***Microsimulation methods for population projection***, Population 10(1), 97–136. [PDF]
- Willekens, F. (2001), ***Theoretical and Technical Orientations Toward Longitudinal Research in the**

Social Sciences*, Canadian Studies in Population 28(2), 189-217 [PDF]

1.3. Population Projections by Micro-simulation: Rationale

Most countries and international agencies produce population projections using the simple cohort-component method. For example, the widely used model DemProj (Stover & Kirmeyer 2001) builds the starting point of the micro-simulation model developed in this report, and it requires:

- A base population table by age and sex for the base year
- Fertility data, including the total fertility rate for the base and future year, and age distribution of women at birth
- A model life table and life expectancy at birth for the base year and assumptions for the future
- Assumptions on net international migration rates

Micro-simulation is a powerful alternative to this approach, as it can overcome technical limitations of macro models. Micro-simulation can not only fully replace the cohort-component method, producing identical results, but also can accommodate step-wise model extensions for added detail. Most importantly, micro-simulation can handle more variables, has no restriction on variable types, possesses memory of individual histories, and allows communication and linkage between people. This makes population projections more useful as they project more characteristics, and can also improve the overall quality of projections. An example is the incorporation of known and very persistent differences in demographic behaviors by specific population groups (e.g., by ethnic affiliation, religion, or income or education level).

A micro-simulation model reproducing a cohort-component model can start off from the same distributional table of the current population, with each person from its starting population sampling its initial characteristics from the distribution table. But as characteristics are added, the feasibility of this approach is quickly reduced, as the number of cells in the table grows exponentially. Micro-simulation overcomes this problem, by reading in a micro-population file (micro-data) as its starting population and allowing it to evolve over time by generating events such as births, marriages, deaths, and migrations.

For the starting population file, one would ideally use the latest population census micro-data or a sample of it (e.g., the five or 10 percent subset that many countries make available for public use). An alternative is to generate a synthetic population, which is a virtual population that resembles very closely the actual population without containing real records, thereby avoiding confidentiality issues.

For fertility, mortality, and migration, the micro-simulation approach also needs data and assumptions. The same information used for the cohort-based approach is sufficient, but we can also include key determinants for demographic events, like education, parity, or geographical context. This would typically use proportional factors, such as relative risks derived in Cox or proportional hazard models, or odds ratios from logistic regression. Required data are typically available in countries that conduct censuses and demographic household surveys like MICS or DHS.

This adds complexity, but it comes with major advantages:

- Parameters can specify population groups
- The use of available information is maximized
- Micro-data allows all kinds of disaggregation, including for small populations, e.g., by ethnic affiliation
- Can test variance because it has a random component
- Simulates the impact of determinants by running different scenarios
- Can plug additional modules in the model (e.g., specific diseases)

A leading application example is Canada’s Demosim model, which projects the Canadian society, including variables like visible minority, Aboriginal identity, education, labor force participation, and a fine-grained geography (Caron-Malenfant & Coulombe 2015). Such projections can use the knowledge of behavioral differences typically found between ethnic groups and separate behavioral changes from composition effects. The added detail of such projections can provide valuable inputs for planning purposes. For example, geographical detail allows for planning of schools and health institutions on a regional level, or for projection of specific population groups, such as Aboriginal peoples. Demosim was also used to assess the impact of potential educational improvements on the future labor force participation of the Aboriginal population (Spielauer 2014). This application provides a simple but powerful example of using micro-simulation for what-if analysis, assessing the impact of policy-induced changes in one behavior (i.e., educational choices) on the outcome of another (i.e., labor force participation); the resulting changes in size and educational composition of the Aboriginal labor force; and the timeline of these changes. Models inspired by Demosim are currently developed for a series of developed countries, including Austria, Germany, and Australia (Belanger, forthcoming). For European countries, the use of micro-simulation for European Union-wide standardized models was explored and demonstrated in the MicMac project (NIDI 2009).

Micro-simulation’s ability to produce detailed population projections is also expected to be of high relevance for applications in developing countries, as they typically experience fast, intertwined demographic and social changes. Demographic behaviors and events (e.g., fertility, child mortality) are often closely linked to policy interventions, e.g., access to higher education or the provision of health care, especially in the developing world.

Although they are the backbone of most dynamic micro-simulation models, demographic modules typically are only one component of micro-simulation models. Models can be very specialized (e.g., modeling specific health trajectories or a specific population group) or, following a modular approach, grow into “multi-purpose” models representing society in many aspects and thus providing a tool for policy-relevant analysis and projections in a wide range of domains.

References

- Caron-Malenfant, E. & Coulombe, S. (2015), ***Demosim: An Overview of Methods and Data Sources***, Statistics Canada Catalogue no. 91-621-X. [PDF]
- Spielauer, M. (2014), ***The relation between education and labour force participation of Aboriginal peoples: A simulation analysis using the Demosim population projection model***, Canadian Studies in Population 41(1-2), 144–164.[PDF]
- Stover, J., S. Kirmeyer (2001), ***DemProj Version 4 - A Computer Program for Making Population Projections***, The POLICY Project Spectrum [PDF]
- NIDI (2009) Bridging the Micro Macro Gap In Population Forecasting. Netherlands Interdisciplinary Demographic Institute. Link to Project: <https://www.nidi.nl/en/research/al/micmac/home>

1.4. A Portable Modular Application

With this report, we developed a customizable modular micro-simulation application for population projections. The model departs from the micro-simulation implementation of a typical macro model; it has a second phase that allows for more detailed population projections and incorporates additional variables and processes, such as the intergenerational transmission of education, first union formation by education, and child mortality by mothers’ characteristics. The model is parameterized for Mauritania, but can be easily ported to other countries, and is implemented using the freely available Modgen micro-simulation language maintained at Statistics Canada. It has a user-friendly graphical user interface with a help function for both the user interface and the model and its modules, parameters etc. Users can easily change parameters and create and save new scenarios. It also has rich table output, which can be

exported to Excel. In addition, the model can write micro-data files, variables to be included, and the points in time for data output chosen by the user. It runs on a standard PC under Windows. The execution time of a model run depends on processor speed, population sample size, time horizon of the simulation, and the user's choice of model selection and alignment routines. A typical model run, starting from an initial population sample of 250,000 persons and a 100-year time horizon, is approximately 10 minutes. This time can increase substantially for some alignment options.

The model is highly modular and contains 11 main modules. Together, they create a fully functional model application, but modules can be replaced and extended or new modules can be added, and the model can be used as a modeling platform. Each module comes with its own data requirements, as discussed in the section 1.5. *Data Requirements*.

This chapter gives a brief overview of the model. A more detailed description of each module, underlying design choices, and the estimation process is given in Section 2 and 3 of this report.

1.4.1. Starting population

The model starts from a starting population file—a standard comma-separated variables (CSV) text file containing nine variables. Records can be weighted and the file length does not have to correspond to the true population size nor the size of the simulated population sample, which are parameters. According to these parameters, when the file is larger than the simulated starting sample, the model automatically samples from the starting population file. If the file is smaller than the chosen starting sample, the model replicates observations. All model output is automatically scaled to the total population size regardless of the chosen sample size for the simulation. Choosing larger samples will reduce Monte Carlo variability at the expense of additional time requirements to run the model.

Parameters:

- The name of the starting population file
- Using weights y/n (can switch weighting off even if the file contains weights)
- The length of the starting population file
- The corresponding total population size
- The size of the simulated population sample

1.4.2. Fertility

Model users can choose between two fertility models, one corresponding to a typical macro model, and the other a refined model that models fertility separately by parity and includes a more detailed list of variables than just age. When using the refined model, the user can choose to align results to the macro model for producing the same aggregated outcomes.

The Base Fertility Module

The base fertility module implements age-specific fertility corresponding to typical macro population projection models. The limitation of this approach is that it ignores important fertility differences, e.g., by number and timing of previous births. Because of this limitation, the model would only produce the right number of children but no realistic female life-courses, even if the future age-specific fertility was known..

Parameters of the base model

- Age distribution of births by calendar year
- Total fertility rate by calendar year

The parameterization of the model can easily change the scenario of the projected period of total fertility rates (TFR) without having to change the age profile of fertility. Internally, the model automatically calcu-

lates fertility rates by age and period.

The base fertility module is used in two alternative ways: as the model to be used to implement fertility, and as the benchmark model. In the latter case, it is used to produce the number of births to which the more detailed refined fertility model can be aligned.

The Refined Fertility Module

Besides age, the refined fertility module models fertility by parity, educational attainment, and union status. First births are parameterized with separate age-specific fertility tables by education and union status. Higher-order births are modeled by proportional hazard regression models estimated separately by birth order. Models contain a baseline risk profile by duration since the previous birth and the relative risks for age group, education, and union status. Additionally, users can create scenarios for future trends.

Model selection:

- Run the “macro” base fertility model
- Run the refined fertility model without alignment
- Run the refined fertility model, but align the total number of births to the macro model
- Run the refined fertility model, but align the total number of births by age to the macro model

Parameters of the refined model:

- First birth rates by age, separately for three educational attainments and two partnership statuses
- Higher-order births: baseline risks by the duration since previous birth and relative risks by age group and educational attainment, separately for births 2 through 15
- Trends by calendar year and birth order

1.4.3. Mortality

For mortality, we provide two versions: a base version that resembles a typical macro model, and a refined model version. The refined model focuses on infant mortality by mother’s characteristics. When using the refined model, the user can align results to the macro model to produce the same aggregated outcomes in the number of deaths for an initial year. After this year, the user can set specific trends for child mortality or use the overall trends of the macro model for all ages.

The Base Mortality Module

Mortality is modeled by age and sex. Parameters are a mortality table (for age patterns) and projected period life expectancy. Within the application, the life table is scaled automatically for each year to meet the targeted life expectancy by calendar year and sex. If no national tables are available, the separation of age pattern and aggregated outcome in life expectancy supports easy scenario building and use of regional standard mortality tables.

Parameters:

- Life table of mortality risks by age and sex
- Life expectancy for projected years by sex

The Optional Child Mortality Module

This module focuses on mortality of children age 0 through 4. In addition to baseline mortality risks by age and sex, relative risks for age of mothers and education of mothers are used. When switched on, this module “overwrites” the base mortality model for children up to their fifth birthday. While this will typically alter the overall life expectancy, the user can align aggregate outcomes for an initial year; in this case,

life expectancy is the same as in the initial year and future differences can be attributed to the changing composition of age and education of mothers. Additionally, the user can set specific time trends by age for child mortality, different from the overall trend (which is calculated automatically to meet the life expectancy parameter).

Model selection:

- Model Choice
- Run the “macro” base mortality model for all ages
- Run the refined child mortality model without alignment
- Run the refined child mortality model with alignment for an initial year
- Choice of trends in child mortality
- Use the overall “macro” trend for all ages
- Use specific trends by age for children 0 through 4

Parameters:

- Base mortality by age and sex for ages 0 through 4
- Relative risks by mother’s education and age group

1.4.4. Internal Migration

Internal migration follows a typical macro approach based on age-specific origin-destination matrices. The user can choose between a base version (by age group and sex) and a refined version, which adds education as a model dimension.

The Base Internal Migration Module

Interprovincial migration is modeled by age group and sex. For easier scenario creation, probabilities to leave a province and distribution of destination provinces by age group and origin are parameterized separately. It is assumed that migration pattern stay constant over time.

Parameters:

- Probabilities to leave a province by age group and sex
- Distribution of destination provinces by province of origin, age group, and sex

The Refined Internal Migration Module

Education is an additional dimension added to the probabilities to leave a province.

Model selection:

- Turn internal migration off
- Use the base module for internal migration
- Use the refined module for internal migration

Parameters:

- Probabilities to leave a province by age group, education, and sex

1.4.5. Immigration

The immigration module implements a typical macro approach, specifying the number of future immigrants, their age distribution, and the distribution of the province. For immigrants, only their sex, place of residence, and age is initially known in the simulation. Other characteristics, including parity, education, union status, and time of last birth, are sampled from the foreign-born population of the same known characteristics, or the total resident population if no donors are found.

Model selection:

- Switch immigration on/off

Parameters:

- Total number of immigrants by sex and calendar year
- Age distribution of immigrants by single year of age, separately by sex
- Distribution of destination provinces by sex and age group

1.4.6. Emigration

The emigration module implements a typical macro approach driven by age-specific emigration rates. It is assumed that emigration pattern stay constant over time.

Model selection:

- Switch immigration on/off

Parameters:

- Emigration rates by age, sex, and province

1.4.7. The Primary Education Module

The modeling of education currently focuses entirely on primary education, a key policy concern in Mauritania. From a modeling perspective, this allows for the decision on education outcome to be modeled early in life, before other processes for which education is used as independent variable. Education is a key variable for first union formation, fertility, child mortality, and migration. At its core, the model is parameterized by probabilities to graduate from primary school by year of birth, province of birth, and sex, respectively. In addition, the model can introduce inter-generational transmission of education by specifying proportional factors (odds ratios) by mother's education. If this option is chosen, the aggregated educational outcome is automatically calibrated for a chosen year of birth to the overall probabilities. In such scenarios, the future dynamics are entirely driven by the changing educational composition of mothers. These model capabilities allow educational change to be separated into changes stemming from inter-generational dynamics, from inter-provincial migration, and from overall trends.

Model selection:

- Including mother's education as relative factor affecting education choices

Parameters:

- Primary school entry age
- Primary school graduation age
- Start of the school year

- Probabilities to enter primary education by year of birth, province of birth, and sex
- Probabilities to complete primary education by year of birth, province of birth, and sex
- Relative influence (log odds) of mother's education on the two educational choices
- First year of birth for which mother's education is added as relative factor to the model and for which the model is calibrated to the overall probabilities

The layout of parameters was chosen to be as intuitive and generic as possible while allowing alternative ways to derive the parameters. For Mauritania, we used a proportional model that can distinguish general trends from inter-provincial differences, which were found very persistent. This allows for both intuitive and alternative scenarios, e.g., persistent inter-provincial differences versus convergence scenarios.

1.4.8. First Union Formation Module

Changes in the age of first union formation is one of the key mechanisms behind fertility changes and many developing societies currently experience a rapid increase in that age, partly resulting from educational expansion. The module for first union formation implements this process in two alternative ways: first by using a parametric Coale & McNeil model, and second by union formation risks by age and education. The Coale & McNeil model uses a very intuitive parameterization with three parameters by year of birth and education: (1) the earliest age of union formation observed, (2) the average age at first union formation, and (3) the proportion of persons ever entering a union. Alternatively, model users can specify age-specific rates of union formation. Such rates can also be derived from the Coale & McNeil model, which produces corresponding table output of the internally calculated rates. This may be useful for policy scenarios that set minimum marriage ages. First union formation is modeled only for females.

Model selection:

- Coale & McNeil model
- Model based on age-specific rates

Parameters:

- Option 1, Coale & McNeil model: minimum and average age of first union formation and proportion of persons entering a union by year of birth and education
- Option 2: Age-specific first union formation ages, by year of birth and education

1.4.9. Running the Model

The model's graphical user interface can edit parameter tables, create and save new scenarios, run the model, and view table output. Table results can be exported into Excel individually or collectively to an Excel Workbook.

- Parameters are organized in tables that can be accessed by clicking on the table name in the navigation area of the application (the left side of the interface). For easier navigation, the list of tables is grouped by topic.
- Clicking on a table name opens the table on the right side of the interface. Tables can be edited directly by the user.
- The application offers scenario control by saving all simulation results with all parameters. Users can create new scenarios by editing parameters and saving the scenario under a new name.

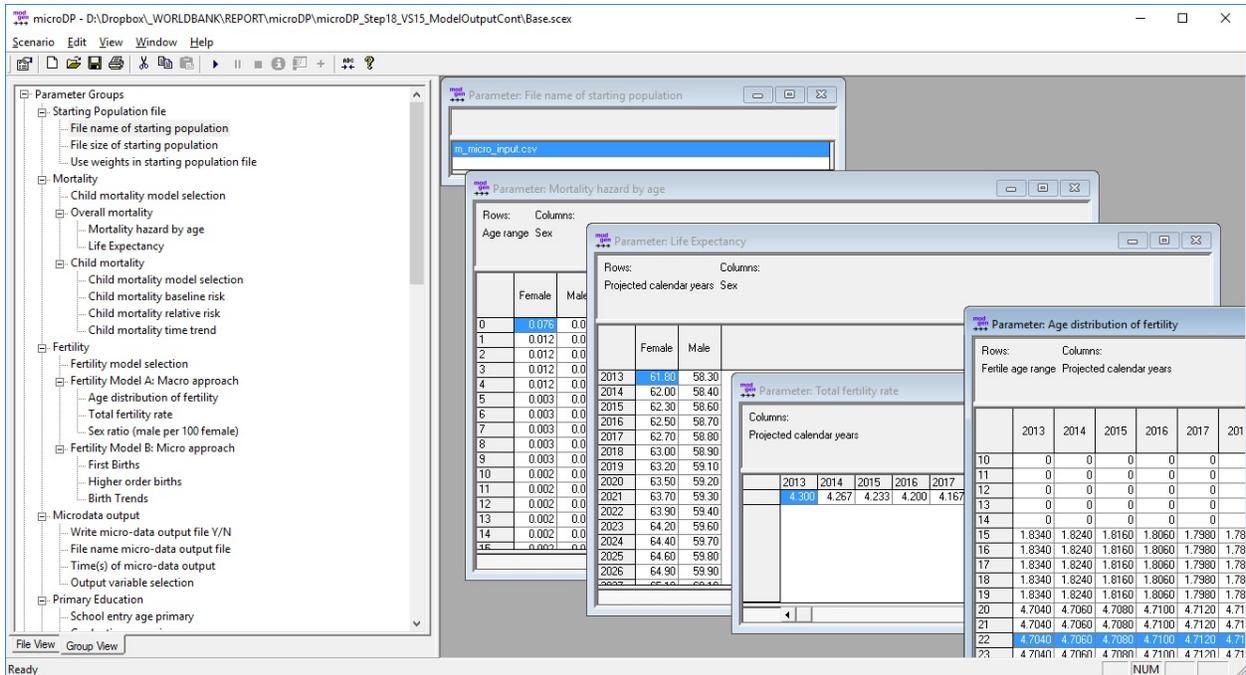


Figure 1-1: The User Interface, Parameter Tables

This screenshot shows the selection of parameter tables essential to the base version of the model: the file name of the starting population, a mortality table, life expectancy, age-specific fertility, and total fertility rate. The user interface is fully documented within the application. The menu offers users access to a detailed hyperlinked help option, which covers relevant aspects, such as editing parameters, creating scenarios, options for running the model, and viewing and exporting model results.

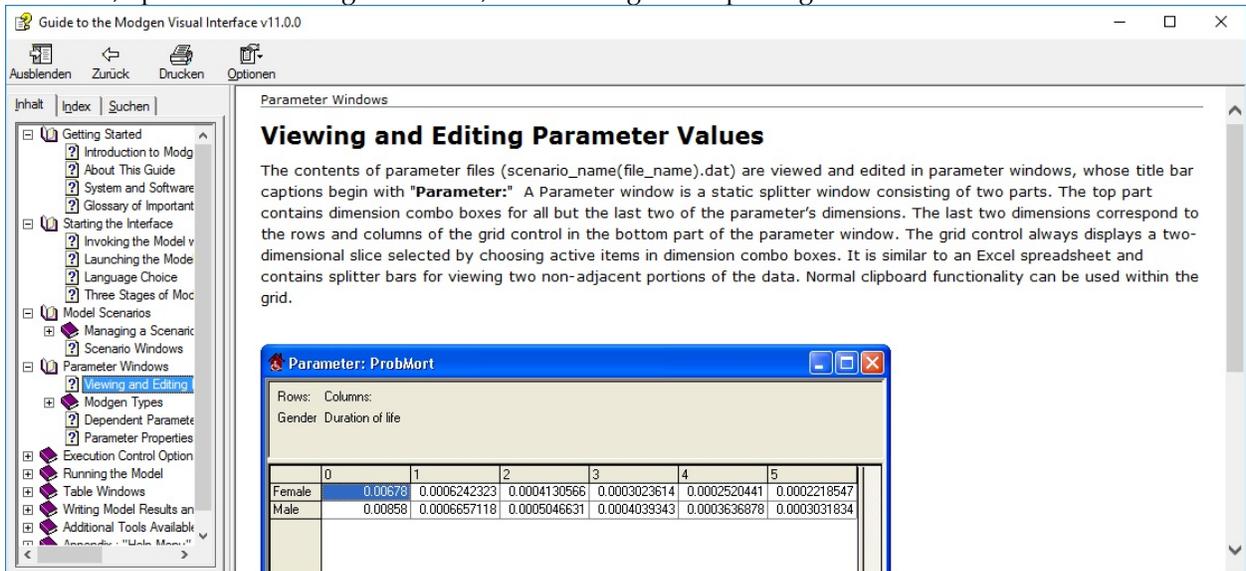


Figure 1-2: The Help Option

Like the user interface, the model is also fully documented within the application. Users can access encyclopedic documentation from the help menu, including descriptions of the modules, parameters, model actors, and all table output.

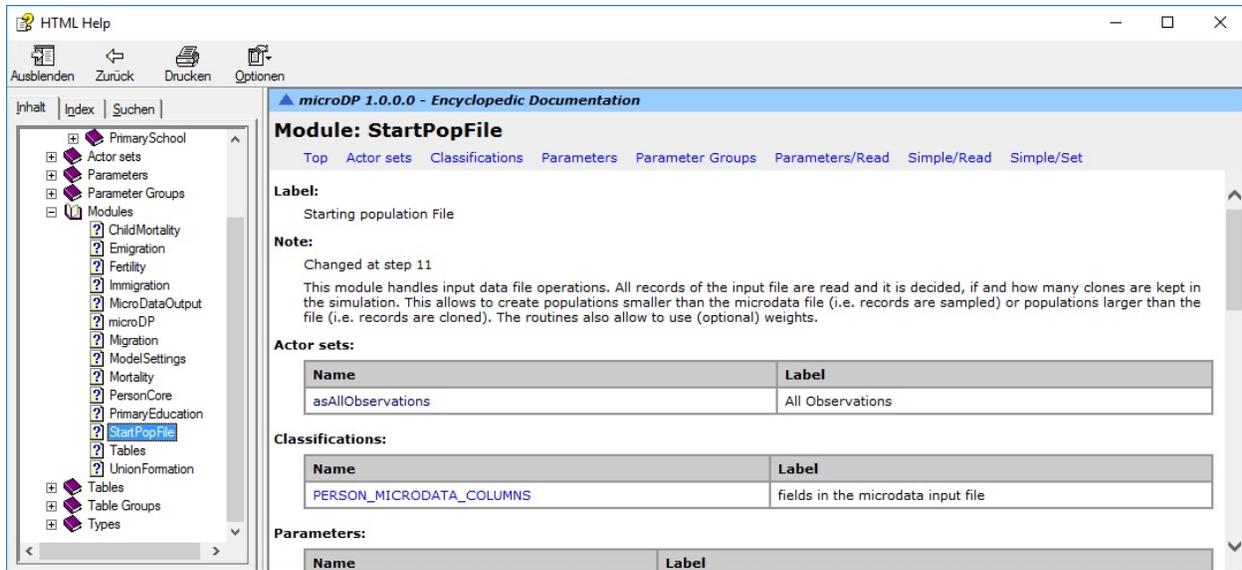


Figure 1-3: The Model Documentation System

- In addition to model parameters, the user also controls some scenario settings. Most importantly, users can choose the time horizon of the simulation and number of replications simulated.
- When running more than one replicate, all model results are automatically calculated as averages over the replicates and distributional information (e.g., the coefficient of variation) is automatically available for each output table cell. This allows users to assess Monte Carlo variation in results.
- The model produces two types of output: a collection of tables and micro-data files for selected moments in time. All tables are updated when running a simulation.
- Tables can have any number of dimensions. For example, population numbers can be displayed by age, year, sex, and province. The user controls how tables are displayed, e.g., a table by age group and year for selected province and sex, or by age group and province for a selected year and sex.
- Like parameter tables, output tables can be opened by clicking on the name in the navigation pane list. For easier navigation, output tables are grouped by topic.

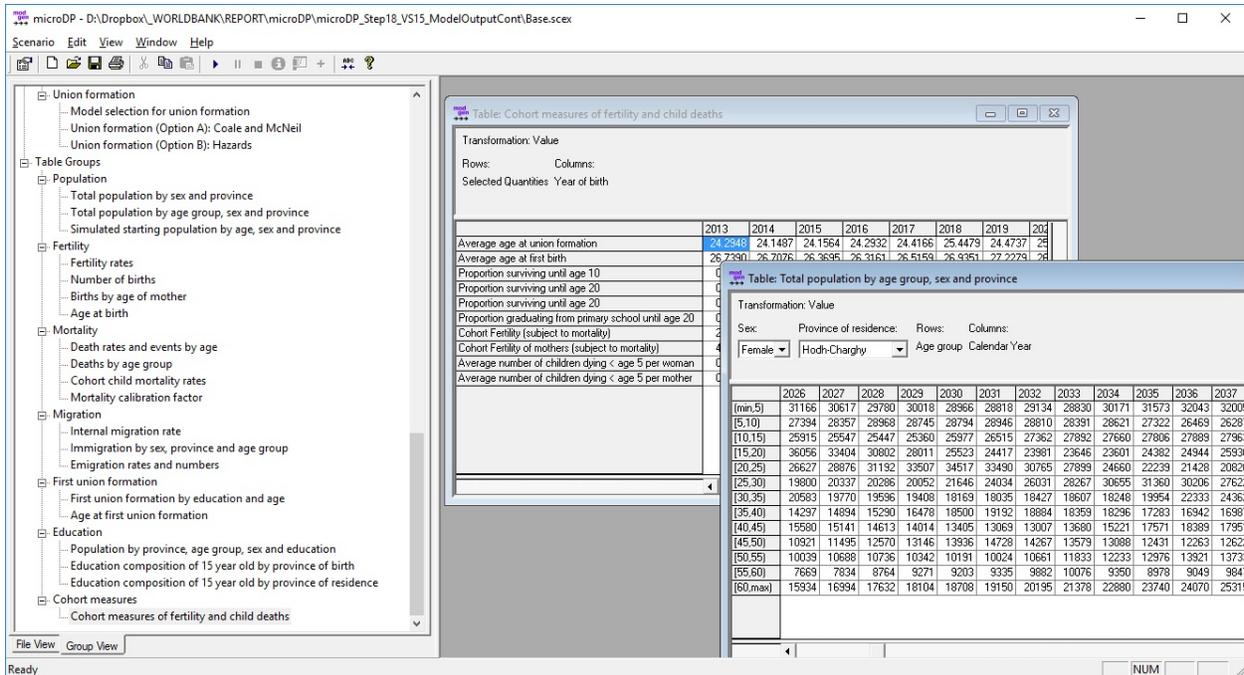


Figure 1-4: Output Tables

1.4.10. Model Output

The model produces a series of tables organized in seven groups. The list of tables can be easily extended by user demand.

Population

- Total population by projected year, sex, and province
- Total population by projected year, age group, sex, and province
- Simulated starting population by age, sex, and province

Fertility

- Age-specific fertility rates by projected year
- Number of births by sex and projected year
- Number of births and first births by age group of mother and projected year
- Average age at birth and at first birth by education and projected year

Mortality

- Death rates and number of deaths by sex, age, and projected year
- Death rates and number of deaths by age group and projected year
- Child mortality by birth cohort and single year of age 0 through 4
- Mortality trends by sex (trend factors calculated internally to scale the standard life table to meet the given scenario of future period life expectancy)

Migration

- Internal migration rates by age group and sex

- Number of immigrants by sex, age group, and province of destination by simulated year
- Emigration rates and number of emigrants by age group, sex, and province by projected year

Union

- First union formation by education, age, and simulated year; rates and proportion of women ever in a union
- Average age at first union formation by education and projected year

Education

- Population by province, age group, sex, and education by projected year
- Education composition of 15-year-old by sex, province of birth, and projected year
- Education composition of 15-year-old by sex, province of residence, and projected year

Female life-course experiences: cohort measures on own survival, union formation, education, fertility, and child deaths

- Average age at union formation
- Average age at first birth
- Proportion surviving until age 10
- Proportion surviving until age 20
- Proportion graduating from primary school
- Cohort Fertility (subject to mortality)
- Cohort Fertility of mothers (subject to mortality)
- Average number of children dying < age 5 per woman
- Average number of children dying < age 5 per mother

1.5. Data Requirements

1.5.1. Overview

Micro-simulation is typically associated with high data demands. This view stems from the predominant use of micro-simulation for modeling highly complex systems, e.g., the operations of social insurance systems in the context of social and demographic change. In contrast to such models, which must depict individual life-courses in great detail and include educational choices, employment, earnings, family dynamics, savings, health, and retirement decisions, the data demands for population projection models are very modest and, for most countries, required data are readily available.

Required data for population projections are life tables for modeling mortality, origin-destination matrices for the modeling migration, and age-specific fertility rates. Here, micro-simulation does not differ from cohort-component models that currently dominate population projections, but the power of micro-simulation is obvious when adding variables and detail. In our application, these variables are primary education, first union formation, and parity—characteristics that allow for a more detailed modeling of the core demographic events. In addition to age, fertility can now account for education, partnership status, and the number and timing of previous births, resulting in the creation of realistic individual life-courses. Concerning mortality, we add a specialized module for child mortality by mother's characteristics (age and education), and migration now adds education as explanatory variable. As a consequence, the

micro-simulation model uses more of the characteristics contained in typical census data, such as education, and will usually require additional information from surveys.

In a nutshell, the micro-simulation population projection presented in this report consists of 11 components and their related data requirements, which collectively add up to 16 variables. Of these components, five are based on the same data as a typical macro projection model:

- **Mortality** is based on a life table and projected changes of life expectancy. Parameters for mortality are usually available from published data. If no reliable national life table of mortality risks by single year of age is available, a standard life table for the world region can be used. The life table is used for age differences in mortality and automatically calibrated (i.e., re-scaled) to meet the second parameter of the model, life expectancy. Future life expectancy is scenario based.
- **Internal migration** is based on origin-destination matrices by age group and sex, which can be tabulated from census data. In an optional model extension, education level is added as another dimension; data are typically available from a population census. For easier parameterization and scenario building, the information is split into two tables: (1) the probability to migrate by age group, sex, and province, and (2) the destination of migrants by province of origin, sex, and age group. In the Mauritanian example, the probabilities to leave are modeled using logistic regression assuming the same age patterns in each province. Depending on sample size, these models can be replaced by cross-tabs when running the analysis on the full census. The transition matrices are produced by cross-tabs. In an optional extension, the same model can be parameterized by education group. The education variable is typically available from census data.
- **Immigration** requires projected total numbers and age distributions by destination province and sex, which can be based on recent numbers, observed in the census, and by applying a time trend.
- **Emigration:** The emigration module orients itself on typical macro population projection models requiring a single parameter table, namely emigration rates by age group, sex, and province. In the Mauritanian case, emigration was modeled from census information collected on household members who left the country in the past 12 months; there were no other data sources.
- **Fertility** is based on age-specific fertility rates and projected trends. Current rates can be tabulated from the census. The future age distribution and total fertility rate are scenario based.

The remaining six components require additional information. In the Mauritanian case, three of these models can be estimated from census data alone:

- **First union formation** uses a parametric (Coale & McNeil) model fitted from census data. The model is parameterized by the earliest age of union formation, average age, and proportion of females eventually entering a union. The parameters are by year of birth and education. Based on the observed proportion of women who have ever entered a union and the age distribution of those observed union formations, curves are fitted that allow projections into the future.
- At its core, the **primary education** module is based on census data, i.e., the highest level of schooling attended and the highest diploma obtained. An optional extension accounts for mother's education. The required proportional factor was estimated from MICS.
- The **refined version of internal migration** uses education as an additional dimension.

The remaining three components require information from survey data. In the Mauritanian case, all additional information is available in MICS data:

- The simulation starts from a micro-database—the **starting population**—which contains nine variables. In the Mauritanian case, all variables but one are available from the census. The file can be the total population or a sample. The most robust available data should be used for the starting population; typically, this would be a population census, a sample thereof, or its synthetic reproduction. Some variables may have to be input from other sources; in the example of Mauritania, a variable was the time of last birth.

WEIGHT	person weight	e.g., 8.275
BIRTH	time of birth	e.g., 1966.532
SEX	sex	0 Female, 1 Male
POR	province of residence	0 .. 12
EDUC	primary education	0 never entered, 1 dropout, 2 completed
POB	province of birth	0 .. 13, (13 for is abroad)
UNION	time of first union (marriage)	e.g. 1988.234
PARITY	number of children ever born	0, 1, ..
LASTBIR	time of last birth	e.g., 2012.34

- This additional variable of time at last birth is also used in the **refined version of fertility** module. Higher-order births were estimated from MICS data, which contain retrospective birth histories that can be used to estimate past time trends. Except for trends, the same models could be estimated from the starting population data set, so they could be mostly based on census data.
- The optional **infant and child mortality** module requires retrospective birth histories and information of deaths as collected from mothers in the MICS survey. Besides an age baseline of child mortality, the model uses relative risks by mother’s characteristics, namely age and education. Estimating these risk factors requires survey data, like the MICS in the Mauritanian case. If not available, relative risks can be borrowed from comparable countries or based on research literature.

MODULE	POPULATION	REQUIRED FOR CORE ANALYSIS											OPTIONAL CORE ANALYSIS					Same as in macro models								
		Weight	Date Birth	Sex	Province of Residence	Province of Birth	Primary Education	Time First Union	Number Children born	Births past 12 months	Time last Birth	Province 12 months ago	Mother's education	Birth Child 1	...	Birth Child 15	Sex Child 1		...	Sex Child 15	Death Child 1	...	Death Child 15	Deceased past 12 months	MORTALITY	
Starting Population	residents	x	x	x	x	x	x	x	x	x	x															
Fertility Model A	female residents	x	x							x																x
Fertility Model B	female residents	x	x		x			x	x	x	x															
Internal Migration Model A	residents	x	x	x	x							x														x
Internal Migration Model B	residents	x	x	x	x			x				x														
Immigration	residents	x	x	x	x							x														x
Emigration	residents and recent emigrants	x	x	x	x							x														x
Primary Education	residents	x	x	x		x	x																			
First Union Formation	female residents	x	x					x	x																	
Child Mortality	female residents	x	x		x	x																				
Mortality	residents and recent deceased	x	x	x																						x
Typical Data Sources	Census	x	x	x	x	x	x	x	x	x	x	x														x
	Surveys	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
	Administrative, standard life tables		x	x	x																					x

Figure 1-5: Components, Variables, and Data Sources of the Micro-simulation Application (x required; o optional)

1.5.2. Data Files

Population and Housing Census 2013

Most analysis is based on a population census. The following variables and files from the Mauritania 2013 Population and Housing Census (Office Nationale de la Statistique (ONS), Recensement Général de la Population et de l’Habitat (RGPH) 2013) were used for the parametrisation of the model:

CENSUS VARIABLES

- M_WEIGHT	Weight
- M_AGE	Age
- M_MALE	Sex
- M_POB	Province of birth
- M_POR	Province of residence
- M_PPROV	Province 12 months ago
- M_EDUC	Education (no - enter - finish primary)
- M_AGEMAR	Age at first marriage
- M_PARITY	Number of births
- M_BIR12	Births in past 12 months
CENSUS FILES:	
- m_census_residents.dta	A file of residents
- m_census_emigrants.dta	A file of recent (past 12 months) emigrants
- m_census_all_incl_emigrants.dta	The complete file including recent emigrants
- m_census_immigrants	A file of recent (past 12 months) immigrants

The file of recent immigrants was generated from the file of residents and used to adjust immigration to better reflect the number and composition of expected future immigrants. In particular, a large number of recent immigrants were refugees from Mali, an immigration flow that will not continue in the future, according to local experts. Some recent immigrants from Mali were therefore removed from the file.

Emigration was modeled from a file of recent emigrants provided from the National Statistical Office. This file was created from census information on household members who left the country and ignores emigrants who did not leave household members in the country. Other information might be more appropriate and available when porting the model to other countries.

All census analysis files were generated from two samples of raw census data, one on the resident population, the other on recent emigrants.

Multiple Indicator Cluster Survey (MICS) 2011

The Mauritania MICS 2011 (Office Nationale de la Statistique (ONS), Enquête par Grappes à Indicateurs Multiples 2011) was also used for analysis. For higher-order births, an analysis file with the following variables was available or created:

File: m_mics_fertility.dta	
Variables:	
- M_ID	Person ID (women)
- M_BIRTH	Birth date (women)
- M_WEIGHT	Record Weight
- M_Bnn	Birth dates of children nn 01-14
- M_EDUC	Primary education: never entered / dropout / finish level 5+
- M_MAR	Date of first marriage

Stata file: 11_MICS_CreateAnalysisFile.do

For child mortality, an analysis file with the following variables was available / created:

File: m_mics_childmortality.dta	
Variables:	
- M_BIRTH	Date of birth (months since 1900)
- M_DEATH	Date of death (months since 1900)
- M_MALE	Sex (Male = 1 / Female = 0)
- M_WEIGHT	Weight

- M_AGEMO	Age of mother at birth (in months)
- M_EDUCMO	Primary education of mother (0 never entered, 1 dropout, 2 graduate)
- M_INTERV	Date of interview (months since 1900)

Stata code: **15_MICS_CreateAnalysisChildMortality.do**

Chapter 2. Phase 1: Reproducing a Typical Macro Population Projection Model

In this and the following section, we develop and describe a two-phase micro-simulation model for population projections. In the first phase, we implement a typical cohort-component macro-projection model, DemProj, as a micro-simulation model. In the second phase, we exceed the capabilities of macro models by adding additional processes, variables, and features and demonstrate some of the key strengths of micro-simulation. The purpose of this exercise is fourfold:

- To provide a projection model for Mauritania with policy relevance in key priority development areas such as child mortality, education, and the management of migration flows
- To provide a portable model, which can be customized for other countries and adapted or refined for additional applications
- To demonstrate the usability and versatility of such a model from a user's perspective
- To provide a step-by-step manual covering key modeling aspects, parameter estimation, and application programming

Components of a population projection are a representation of the current population—e.g., a micro-data file or a distributional table—and a model for moving this population forward. Key elements replicated from DemProj at this step are models for fertility, internal migration, international migration, and mortality. This model will be further developed in the next section by adding modules for education, first union formation, and child mortality, and by refining the modules for fertility and migration.

2.1. The Base model: Reproducing a macro model

We chose to start the development of a population projection micro-simulation model by reproducing the typical macro model DemProj. While such an exercise does not immediately add anything to DemProj, there are benefits to this approach:

- We demonstrate that a micro-simulation implementation of a population projection model based on the cohort-component method is a relatively straightforward process and an alternative way to implement the same model.
- We help lower the entry barrier to run micro-simulation models from a user's perspective by providing a familiar model with parameter tables and results identical to the DemProj macro model. The model can be used as a training tool to introduce features of the user interface common to all Modgen models.
- By producing projections that are identical to published macro projections, the model produces benchmarks that can also be useful in the following model refinements and steps, both for comparison of results and as alignment targets. In the latter case, micro-simulation is used to add detail to

projections (e.g., a realistic distribution of family sizes by education) while reproducing aggregate outcomes (e.g., number of births) as projected by cohort-component models.

- From a technical perspective, the micro-simulation implementation of DemProj introduces most key concepts of micro-simulation programming and the modules can be used as typical building blocks of micro-simulation models.
- In addition to all the demonstrative reasons listed above, the micro-simulation implementation of DemProj also provides a good and logical starting point for model extensions beyond the capabilities of macro models. It already implements the core modules of a population projection model based on the population representation by individual actors, which can be easily refined by adding individual-level characteristics, additional events, and communication and interactions between actors.

The data needed in this phase are identical to those of a macro population projection model. In the case of Mauritania, scenarios for fertility and mortality were supplied by the National Statistical Office (ONS). Data sets for individual countries, including Mauritania, and regions can also be downloaded from DemProj [project sites](http://www.healthpolicyproject.com/index.cfm?id=software&get=Spectrum) (e.g., <http://www.healthpolicyproject.com/index.cfm?id=software&get=Spectrum>). The ONS currently does not include international migration in its macro projections, and provincial distributions are input into national projections. In contrast, we explicitly include internal migration, emigration, and immigration in the micro-simulation. (All corresponding models can be disabled by users.) Current measures for internal migration, immigration, and emigration were calculated from a sample of census data. The estimation of approximate emigration was based on census records of household members who recently left the country.

2.2. Starting Population

2.2.1. Discussion

DemProj, as a cohort-component model, is cell based. Cells are defined by all possible combinations of individual characteristics, such as age, sex, and province, and the model's population information is represented by the count of people in each cell, i.e., a table of persons by age, sex, and province. Technically, the same approach can be used to start a micro-simulation model; there are two possible ways: directly, producing one person or a multiple of clones for each person on the table, or using the table to sample the characteristics of simulated actors. The latter approach is more common and more flexible, as it allows the user to choose the sample size. It is also typically sufficient to run a sample of a population and allows scaling of the model.

This approach has the same limitations as all cell-based models: by adding characteristics, the number of cells explodes exponentially. For example, a model by single age 0-100, sex, 13 provinces of residence, 14 provinces of birth (including abroad), 6 education groups, a range of children (parity) 0 - 15, and 4 categories of marital status would have $101 \times 2 \times 13 \times 14 \times 6 \times 16 \times 4 = 14.12$ million cells—in our case, a number four times the population of Mauritania. Cell-based models would have to identify all possible transitions between each cell and model transition probabilities, an unfeasible task that limits the number of characteristics that can be handled by this approach to very few.

From a micro-simulation perspective, representing a population by multidimensional distributional tables is useful only if the storage space of such a table does not exceed a micro-data file containing the same information. In addition, a cell-based representation of a population does not allow continuous variables, e.g., income, and would have to split them into categories. Using a micro-data file to start a simulation is thus the most common approach in micro-simulation, which is typically used for detailed modeling. The exception to this rule—complex models starting from distributional tables—are so-called synthetic models, such as the Canadian LifePaths. In this case, all persons are created with very few characteristics

sampled from distributional tables, and other characteristics are modeled from birth over the life-course, starting the simulation in the past. Seen as an imputation technique, modeling characteristics longitudinally can be used as a micro-simulation application by itself (reconstructing the past) and models for the past can be kept to project characteristics into the future (if it adequately reproduces the past). A second way for adding missing characteristics consists of cross-sectional imputations at the beginning of the simulation. Cross-sectional imputations are usually performed outside of the micro-simulation model by tools like SimPop, which was used to prepare the starting population file of our application in its second development step. Here, in a first step, the starting population is represented by a three-dimensional table of the population by age, sex, and province, with information from the 2013 census.

Another design choice in micro-simulation is when to start the simulation of each actor. This can happen in the past, thus creating all actors with age 0 and aging them until the starting point of the simulation, or creating actors at the starting moment of the simulation, with the age and all characteristics at this moment in time. The first approach allows the type of longitudinal imputations of characteristics described above and is thus more flexible in giving people individual histories. In our case, given the simplicity of the model, both approaches work. To demonstrate the implementation of both approaches, we create all persons of the starting population at age 0, but create future immigrants “on demand” at the moment of immigration. While the former approach in Step 2 will also be used to impute some past information, such as school graduations, the latter is used to demonstrate cross-sectional imputations of missing information, e.g., by sampling missing variables from the resident population or recent immigrants with otherwise identical characteristics.

2.2.2. Parameters

The starting population of the base version of the model is contained in a single parameter table by age, sex, and province.

The screenshot shows a window titled "Parameter: Starting population" with a dropdown menu set to "Female". Below the menu is a table with 14 columns representing provinces and 14 rows representing ages from 0 to 13. The data values are as follows:

	Hodh-Charghy	Hodh-Gharby	Assaba	Gorgol	Brakna	Trarza	Adrar	Jakhlett-Nouadiboi	Tagd
0	6272.40	4361.88	5568.03	5467.45	5898.57	4876.32	1104.61	1845.82	
1	6953.60	4911.63	4587.04	5494.30	4925.70	3857.13	980.90	1451.48	
2	9085.69	6461.72	6185.70	7382.40	5837.22	4630.31	1095.78	1686.41	
3	8846.82	6533.81	6994.11	7113.95	6292.98	4718.17	1139.96	1594.12	
4	9174.15	6732.08	6785.19	7275.02	5451.58	4349.15	786.49	1594.12	
5	7683.82	6070.14	6207.25	6424.35	6071.64	4629.98	833.01	1787.09	
6	7106.75	5094.58	5076.56	5776.45	5041.06	3767.84	793.34	1443.09	
7	7336.00	4839.08	5322.70	5932.57	5272.74	3927.50	904.41	1669.63	
8	6703.58	4289.36	4822.73	5737.42	4873.29	3767.84	753.67	1233.34	
9	4505.95	3337.03	3753.58	4012.29	3467.22	2817.90	674.34	1132.66	
10	7429.14	4716.46	5325.04	6001.41	5455.85	3855.96	937.96	1736.75	
11	4124.54	3213.51	3552.76	3243.56	3060.80	2552.09	577.85	964.86	
12	6592.63	4625.12	5046.07	5112.31	4855.06	3959.31	770.47	1300.46	
13	4348.16	3147.08	3503.53	4198.51	3385.55	2782.65	720.22	1006.81	

Figure 2-1: Starting Population Parameter

2.2.3. Analysis

The starting population parameter is tabulated from the population census file.

- Stata file: 02_StartingPopulations.do
- Excel Parameter Tables: Starting_Population.xlsx

```
/* ***** */
```

```

/* DATA RESIDENT POPULATION */
/* ***** */

clear all
#delimit;
use "Microdata\m_census_residents.dta";

/* ***** */
/* DISTRIBUTIONAL TABLE */
/* ***** */

gen M_NAGE = int(M_AGE);

tab M_NAGE M_POR [iweight=M_WEIGHT] if M_MALE == 0;
tab M_NAGE M_POR [iweight=M_WEIGHT] if M_MALE == 1;

```

2.3. Fertility

2.3.1. Discussion

The fertility module is based on age-specific fertility rates calculated from two parameters: an age distribution of fertility and the TFR for future years. In this parameterization, the model directly follows the DemProj approach, which splits fertility projections into one of age patterns and another of TFRs. The third parameter is the sex ratio.

The quality of the projections relies on the quality of the chosen fertility rates, and a wide body of literature exists for how to project fertility patterns. Typical patterns and trajectories were identified and are typically applied for developing countries, e.g., assuming common development stages of fertility transitions. The macro approach itself is limited to model fertility by age. Age is an important predictor for fertility itself, but also important for modeling maternal and child mortality, as those risks are far higher at both age extremes of the fertile cycle.

The implementation of this approach is straightforward in micro-simulation, where the same parameters of age-specific fertility are used in the macro projection to determine the number of births added to the population each year and are applied to individual women. Births are modeled in continuous time, thus can happen at any moment of time. For each woman age 15-49, a random waiting time to the next birth is calculated from the age and period-specific fertility rates calculated from the parameters. As rates are updated at each birthday and calendar years change, a birth event occurs and a new actor is added to the simulation if the waiting time is shorter than the time to the next update of rates, otherwise a new waiting time based on the new rate is generated, and so forth. At the birth event, the sex of the baby is determined by sampling from the sex ratio parameter. The province of birth as well as the moment of birth are information transmitted from mother to child.

Modeling fertility exclusively by age does not result in a realistic representation of female birth histories and the distribution of family sizes as it ignores important other individual characteristics, like marital status, the current family size, education, and the time since last birth. Some of these characteristics will be modeled and added in the second step of model development, adding both realism and policy relevance to the model.

2.3.2. Parameters

The fertility module is based on age-specific fertility rates calculated from two parameters: an age distribution of fertility and the TFR for future years. In this parameterization, the model directly follows the DemProj approach, which divides fertility projections into projections of age patterns and of total fertility

rates. The third parameter is the sex ratio.

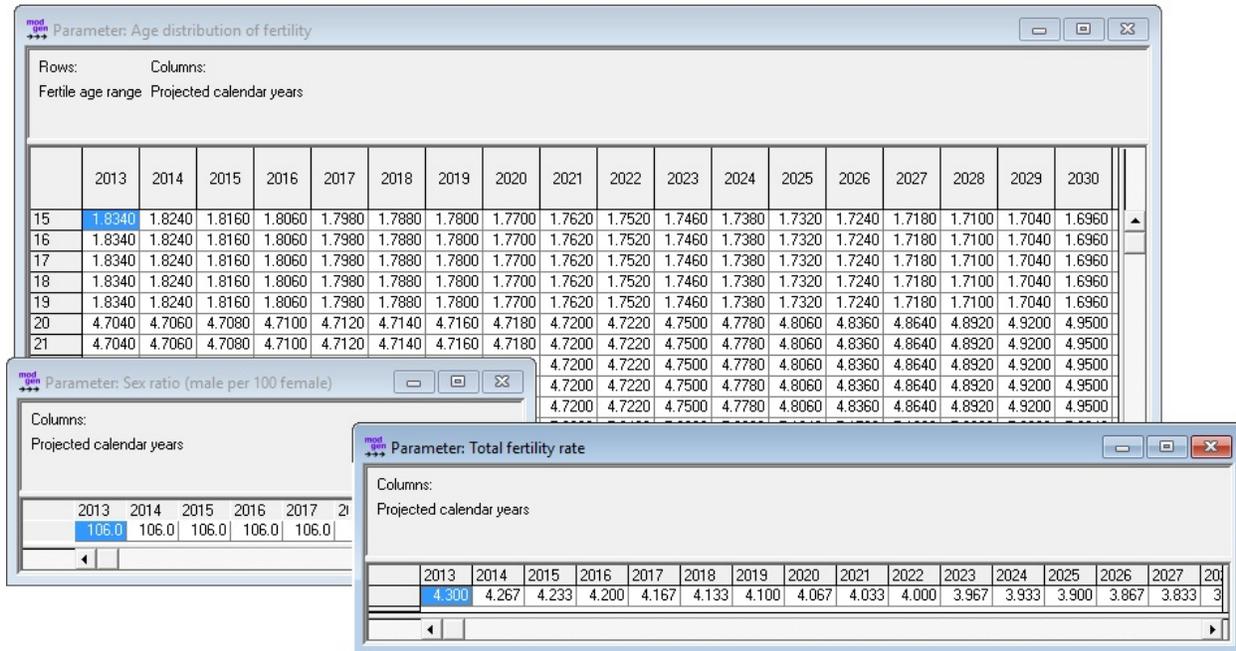


Figure 2-2: Fertility Parameters. (1) Age Distribution, (2) Sex Ratio, (3) Projected TFR

Unlike DemProj, which works with fertility distributions by five-year age groups, the micro-simulation implementation allows parameters by single years of age. While this allows for future refinements and provides more flexibility, the current parameterization reproduces DemProj.

2.3.3. Analysis

All parameters were provided by the Mauritanian ONS for the years 2013 through 2043. From there on, age patterns and sex ratio are kept constant, while total fertility rates are pro-rated, continuing a linear trend until reaching a rate of 2.0, which is kept stable. Data sets for individual countries, including Mauritania, and regions can also be downloaded from DemProj project sites (e.g., <http://www.healthpolicyproject.com/index.cfm?id=software&get=Spectrum>).

Current fertility rates can be estimated from census data if a link between 0-year-old children and their mothers can be established or women report births in the past 12 months. The first approach, the own child approach - assumes that babies stay with their mothers, who can be identified in the household. The approach can be refined by correcting for biases due to mortality. The second approach models fertility from the mother’s side. An example of estimating fertility rates from census data can be found in Chapter 3.4.

- Excel Parameter Tables: Base_Fertility.xlsx

2.4. Mortality

2.4.1. Discussion

Mortality is modeled and implemented following the approach in DemProj, i.e., by working with a standard mortality table for age patterns and a projected period of life expectancy. Within the application, the life table is scaled automatically for each year to meet the targeted life expectancy by calendar year and

sex. Like fertility, mortality is modeled in continuous time, thus death can occur at any moment.

Life tables are central tools in demography, and extensive literature exists in mortality projections. As a consequence, although projections are available from various sources based on various methods, they are typically based on persistent patterns of mortality reduction observed in many countries over the past century. As for other processes, micro-simulation can add detail to mortality projections, e.g., by accounting for the observed and typically very significant and pronounced influence of education, marital status, or, in detailed models, health status and individual diseases. Micro-simulation is especially powerful for modeling inheritable and transmittable diseases e.g., HIV, as it can model interactions between actors.

Note that due to the separate parameterization of the age profile of mortality and life expectancy, changes in age profile alone do not result in changes in overall life expectancy, as the life table is automatically scaled to produce a targeted life expectancy. For example, lowering child mortality in the age profile would increase mortality at other ages to compensate for the effect. In the second step of model development, we allow for scenarios that improve on child mortality without re-scaling mortality at other ages, to “open” the model for the detailed modeling of child mortality by characteristics like mother’s education or parity.

2.4.2. Parameters

The two parameters of the base version of the model are a standard life table and projected life expectancy.

Parameter: Mortality hazard by age

Age range	Female	Male
0	0.076	0.076
1	0.012	0.012
2	0.012	0.012
3	0.012	0.012
4	0.012	0.012
5	0.003	0.003
6	0.003	0.003
7	0.003	0.003
8	0.003	0.003
9	0.003	0.003
10	0.002	0.002
11	0.002	0.002
12	0.002	0.002
13	0.002	0.002
14	0.002	0.002
15	0.002	0.002

Parameter: Life Expectancy

Projected calendar years	Female	Male
2013	61.80	58.30
2014	62.00	58.40
2015	62.30	58.60
2016	62.50	58.70
2017	62.70	58.80
2018	63.00	58.90
2019	63.20	59.10
2020	63.50	59.20
2021	63.70	59.30
2022	63.90	59.40
2023	64.20	59.60
2024	64.40	59.70
2025	64.60	59.80
2026	64.90	59.90

Figure 2-3: Mortality Parameters

2.4.3. Analysis

Data for this model were provided by ONS. Data sets for individual countries, including Mauritania, and regions can be downloaded from DemProj project sites (e.g., <http://www.healthpolicyproject.com/index.cfm?id=software&get=Spectrum>). In general, two sets of model life tables are employed by DemProj: the CoaleDemeny (Coale, Demeny, and Vaughan, 1983) model and the United Nations tables for developing countries (United Nations, 1982). Projections of life expectancy can have various sources: national projections, national goals, UN and U.S. census bureau projections, recent trends and international experience projections, or an application of the UN model schedule of mortality improvements.

Excel Parameter Tables: Base_Mortality.xlsx

2.5. Internal Migration

2.5.1. Discussion

As typical in macro models, migration is modeled using origin-destination matrices by age group and sex. For easier parameterization and scenario building, migration transitions are modeled here in two steps. The first is the probability to leave a province by current province of residence, five-year age group, and sex. The second step is the decision where to move, based on origin-destination matrices for movers by departure province, age group, and sex. While internal migration can be switched off in the model, it otherwise assumes the same migration rates in the future—a typical assumption in population projection models.

Analyses show higher rates of migration for more highly educated people, with different magnitudes by province. In the second modeling phase, the option to simulate migration by education is added. Micro-simulation can be a powerful tool for detailed simulation of migration decisions, both on the individual and family levels. Such models can include contextual information on regions (e.g., public infrastructure, like the availability of schools) as well as regional events like droughts and climate change. From that perspective, the models can be starting points for more detailed, policy-relevant modeling.

2.5.2. Parameters

The base migration module has three parameters: an on/off switch to disable migration; modeling, probability tables to migrate by age group, sex, and province; and the distribution of destination provinces by province of origin, age group, and sex.

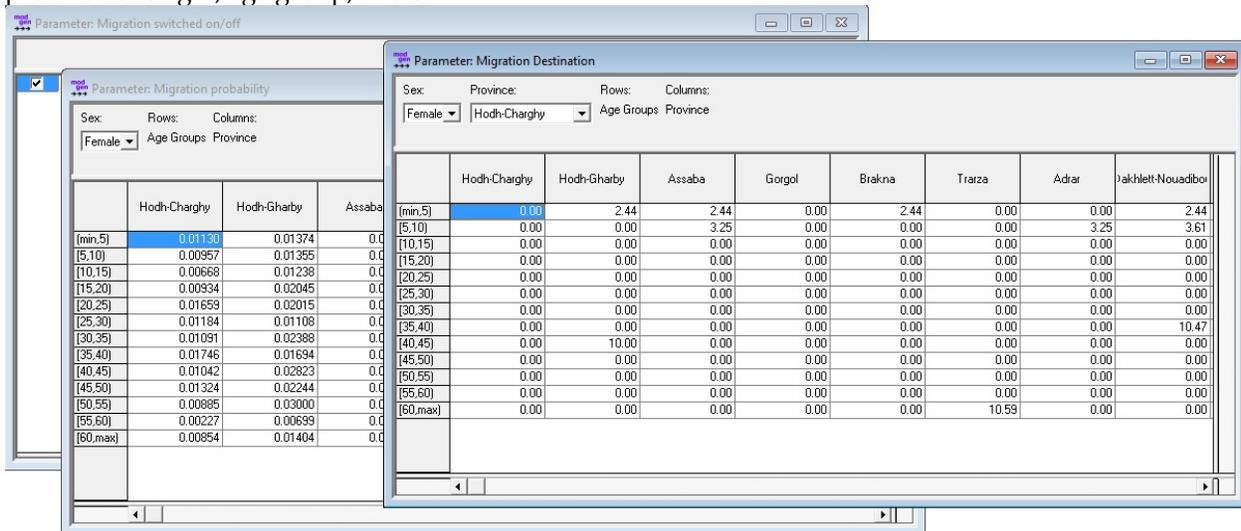


Figure 2-4: Internal Migration Parameters

2.5.3. Analysis: Estimating the Parameters

The ONS of Mauritania currently uses a Demproj version without internal migration; no “ready-made” official migration transition data are currently available. This gap is closed by estimating the model based on a census sample. Due to sample size restrictions, logistic regression is used to estimate probabilities to leave. This assumes a common age profile of migration risks for all provinces, which differ by the same proportional factor over all ages. Depending on age, between 60 and 80 percent of migrants move to the capital Nouakchott, which itself has only low rates of out-migration. Migration probabilities are very

different between provinces; for some provinces and age groups (20-35), they reach or even exceed 10 percent.

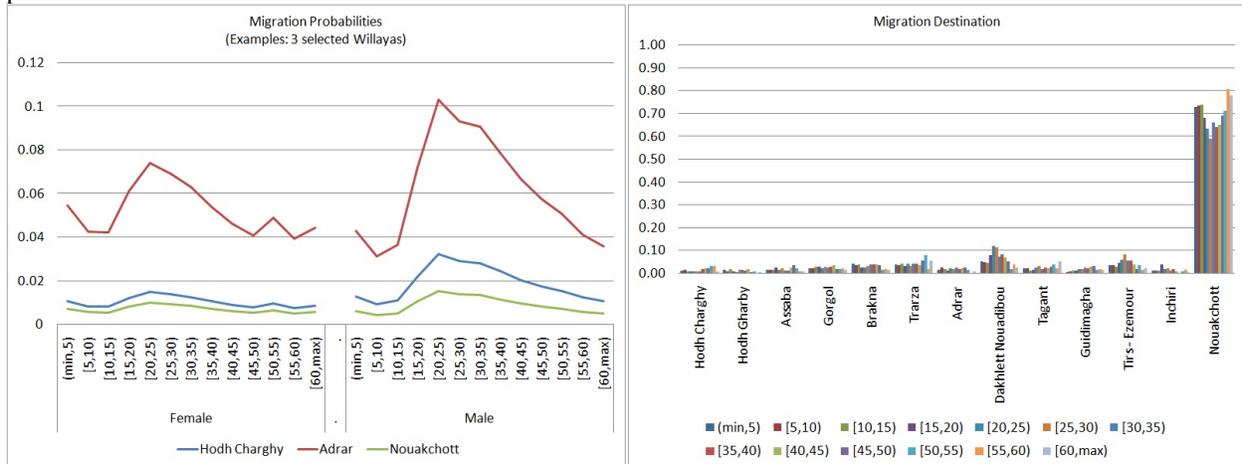


Figure 2-5: Migration Probabilities (3 Provinces) and Destinations

- Stata code file: 03_BaseMigration.do
- Auxillary data: m_empty_migrants.dta
- Excel parameter file: Base_Migration.xlsx

The model of probabilities to leave that currently uses logistic regression can be replaced by cross-tabs when running the analysis on the full census. The transition matrices are cross-tabs. To fill all cells of the transition matrix, a file with very low-weighted movers was created, allotting one person by sex and age group for each possible residential move. The code to append this file can be removed when running the analysis from the full census.

The essential code:

```

/* ***** */
/* DATA: RESIDENT POPULATION */
/* ***** */

clear all
#delimit;
use "Microdata/m_census_residents.dta";

/* ***** */
/* VARIABLES */
/* ***** */

/* Age one year ago, drop if < 0 */
gen m_ageago = int(M_AGE)-1; drop if m_ageago < 0;

/* Age groups 5 years ago, up to 60+ */
gen m_agegr5 = 5 * int(m_ageago / 5); replace m_agegr5 = 60 if m_agegr5 > 60;

/* Person is internal migrant (moved within the country)*/
gen is_migrant = 0; replace is_migrant = 1 if M_PPROV != M_POR & M_PPROV != 13;

/* Append (very) low weighted transition records of all possible transitions to avoid
empty cells */
append using "m_empty_migrants.dta";

```

```

/* Person is potential migrant (not a recent immigrant) -- drop others */
drop if M_PPROV == 13;

/* ***** */
/* MODELS */
/* ***** */

/* Leaving */

xi: logit is_migrant i.m_agegr5 i.M_PPROV [iweight=M_WEIGHT] if M_MALE == 0;
xi: logit is_migrant i.m_agegr5 i.M_PPROV [iweight=M_WEIGHT] if M_MALE == 1;

/* Destinations by origin */

by M_PPROV M_MALE, sort : tabulate m_agegr5 M_POR [iweight = M_WEIGHT] if is_migrant,
    nolab row nofr;

```

2.6. Immigration

2.6.1. Discussion

As is typical for macro models, immigration is modeled by total projected numbers by year, age, and sex distribution of immigrants, and the distribution of destination provinces by age and sex. In the current implementation, only the parameter for total numbers of immigrants has a calendar time dimension. The age and destination distributions are assumed to be time invariant.

Similar to macro population projection models, the base version of the model does not provide immigrants with characteristics other than age and sex, but can be easily added as entities in the simulation. Modeling of immigration can become a complex and interesting issue in more detailed micro-simulation models. In our case—in the second phase—additional characteristics will be education, parity, date of first union formation, and time of last birth. There are alternative ways to create these characteristics, including simulation of immigrants from birth or the cross-sectional imputation of missing characteristics at the moment of immigration. We have chosen a very simple cross-sectional imputation method, consisting of sampling characteristics from the foreign-born host population. Such an approach will be sufficient if numbers of immigrants are relatively low and their characteristics do not differ much over time. The simulation of immigration can become a central focus in population micro-simulation in more detailed models, including immigration policies (selecting immigrants e.g., by education) or refugee migration.

2.6.2. Parameters

The model has four parameters: an on/off switch to allow simulations with and without immigration; total number of immigrants by sex and calendar year; age distribution of immigrants by single year of age and sex; and distribution of destination provinces of immigrants by sex and five-year age groups.

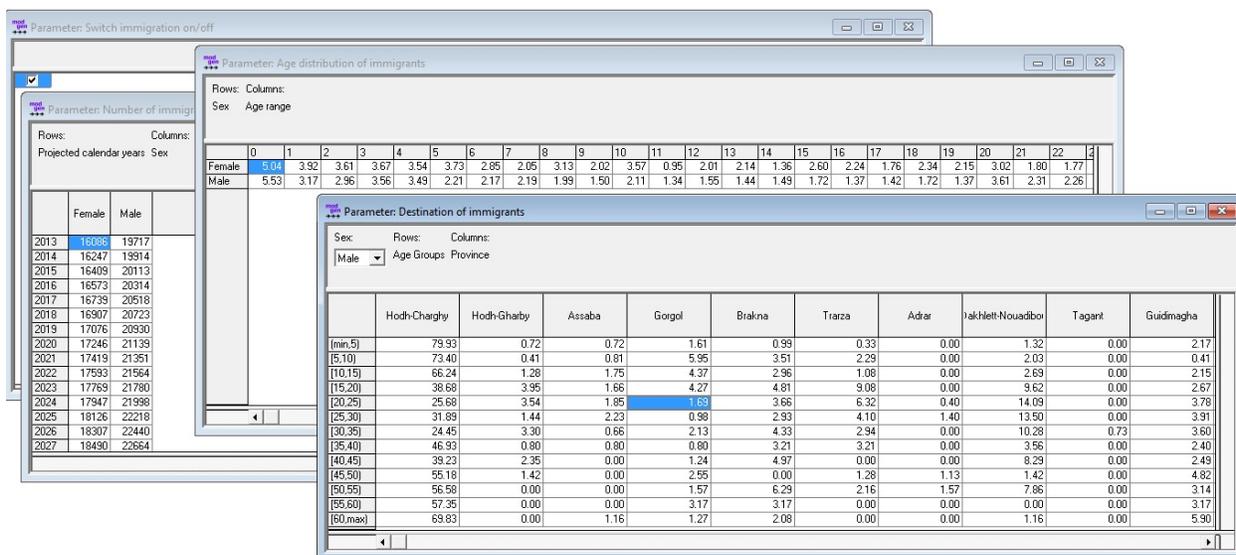


Figure 2-6: Immigration Parameters

2.6.3. Analysis

The ONS of Mauritania currently uses DemProj without immigration and no “ready-made” official immigration data are currently available. This gap is closed by using a 12 percent sample of the 2013 census to model immigration. The immigration module is based on census observations of immigrants having entered the country in the 12 months prior to the 2013 census. Based on estimates from the sample, the total number of immigrants is 37,000. This number contains a high number of refugees from Mali: 72.56 percent female and 63.00 percent male. Of those, 92.15 percent female and 80.20 percent male arrive in Hodh Charghy. According to local experts, it is supposed these immigrants will stay, but no new immigrants from Mali are expected. To account for that, we assumed a 95 percent drop in this immigration group. Such a scenario results in a total number of immigrants of 12,800. There is an interesting difference in the destination of immigrants by age: Nouakchott over-proportionally receives people in the active age range, while very young and older people are over-represented in Hodh Charhy.

Immigrants are defined as those who have lived in the current Willaya less than a year and previously lived abroad. Babies age 0 are treated as immigrants if they were born abroad. For those currently above age 0, a 50 percent probability is assumed, that their full age in years was one year younger at immigration. The model uses simple census cross-tabulations. In the simulation, all immigrants arrive at mid-year.

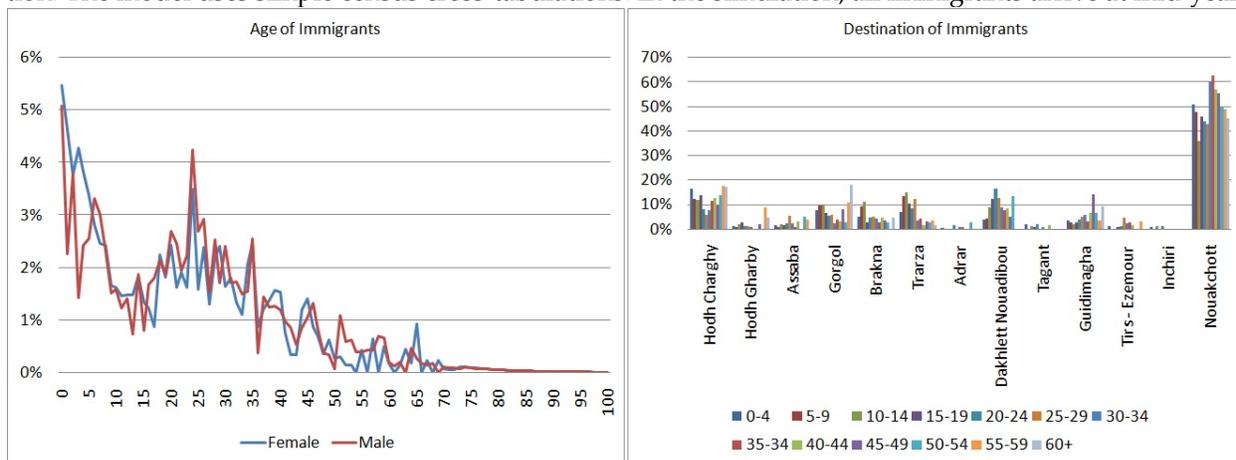


Figure 2-7: Age Distribution and Destination of Immigrants

- Stata code: 05_Immigration.do
- Excel parameter file: Immigration_Module.xlsx
- Auxilliary file: m_empty_immigrants.dta

To fill all cells of the destination matrices, a file with very low-weighted movers was created, allotting one person by sex and age group for each possible destination. The code to append this file can be removed when running the analysis from the full census.

```

/* ***** */
/* DATA: IMMIGRANT POPULATION */
/* ***** */

clear all
#delimit;
use "D:\Dropbox\WorldBank2016_MS\Census\m_census_immigrants.dta";

/* ***** */
/* VARIABLES */
/* ***** */

/* Age at immigration: 50% chance of having been a year younger
   The high number of 0 year old is unrealistic, 50% are given a random age < 10
*/
gen m_age = int(M_AGE);
replace m_age = m_age-1 if runiform()<0.5; replace m_age = 0 if m_age < 0;
replace m_age = m_age + int( 10 * runiform() ) if m_age== 0 & runiform()<0.5;
tab m_age;

/* 5 year age groups, age at immigration, up to 60+ */
gen m_agegr5 = 5 * int(m_age / 5);
replace m_agegr5 = 60 if m_agegr5 > 60;
label define l_agegr5
    0 "0-4"
    5 "5-9"
    10 "10-14"
    15 "15-19"
    20 "20-24"
    25 "25-29"
    30 "30-34"
    35 "35-34"
    40 "40-44"
    45 "45-49"
    50 "50-54"
    55 "55-59"
    60 "60+" ;
label values m_agegr5 l_agegr5;

/* Append (very) low weighted records of all possible destinations to avoid empty
cells */
append using "D:\Dropbox\WorldBank2016_MS\Census\m_empty_immigrants.dta";

/* ***** */
/* VARIABLES */
/* ***** */

/* TAB 01: Number of immigrants by sex */
tabulate M_MALE [iweight = M_WEIGHT];

```

```

/* TAB 02: Destination of immigrants */
by M_MALE , sort : tabulate m_agegr5 M_POR [iweight = M_WEIGHT], row nofr;

/* TAB 03: Age distribution of immigrants by sex */
tabulate m_age M_MALE [iweight = M_WEIGHT], col nofr;

```

2.7. Emigration

2.7.1. Discussion

Emigration is modeled by age-specific emigration rates by province and sex. Because emigration is not a priority of this report, this simple module will be kept in the second phase of model development. Similar to migration and immigration, micro-simulation can be a powerful tool for a detailed modeling of emigration. This may be relevant to developing countries in which emigration is frequent and relatives living abroad may aid resident relatives and contribute to development by remittances. On the negative side, brain drain might constitute a considerable problem for some societies and can be modeled specifically using micro-simulation.

2.7.2. Parameters

In addition to an on/off-switch, the only parameter for emigration is a table of emigration rates by province, age group, and sex.

	Hodh-Charghy	Hodh-Gharby	Assaba	Gorgol	Brakna	Trarza	Adrar	Jakhlett-Nouadibor	Tagant	Guidimgha	Tirs-Ezour
[min,5]	0.00055	0.00055	0.00055	0.00055	0.00055	0.00055	0.00055	0.00055	0.00055	0.00055	0.00055
[5,10]	0.00050	0.00050	0.00050	0.00050	0.00050	0.00050	0.00050	0.00050	0.00050	0.00050	0.00050
[10,15]	0.00064	0.00064	0.00064	0.00064	0.00064	0.00064	0.00064	0.00064	0.00064	0.00064	0.00064
[15,20]	0.00146	0.00146	0.00146	0.00146	0.00146	0.00146	0.00146	0.00146	0.00146	0.00146	0.00146
[20,25]	0.00210	0.00210	0.00210	0.00210	0.00210	0.00210	0.00210	0.00210	0.00210	0.00210	0.00210
[25,30]	0.00168	0.00168	0.00168	0.00168	0.00168	0.00168	0.00168	0.00168	0.00168	0.00168	0.00168
[30,35]	0.00218	0.00218	0.00218	0.00218	0.00218	0.00218	0.00218	0.00218	0.00218	0.00218	0.00218
[35,40]	0.00137	0.00137	0.00137	0.00137	0.00137	0.00137	0.00137	0.00137	0.00137	0.00137	0.00137
[40,45]	0.00162	0.00162	0.00162	0.00162	0.00162	0.00162	0.00162	0.00162	0.00162	0.00162	0.00162
[45,50]	0.00054	0.00054	0.00054	0.00054	0.00054	0.00054	0.00054	0.00054	0.00054	0.00054	0.00054
[50,55]	0.00109	0.00109	0.00109	0.00109	0.00109	0.00109	0.00109	0.00109	0.00109	0.00109	0.00109
[55,60]	0.00024	0.00024	0.00024	0.00024	0.00024	0.00024	0.00024	0.00024	0.00024	0.00024	0.00024
[60,max]	0.00022	0.00022	0.00022	0.00022	0.00022	0.00022	0.00022	0.00022	0.00022	0.00022	0.00022

Figure 2-8: Emigration Parameters

2.7.3. Analysis

- Stata code: 04_Emigration.do
- Excel parameter file: Emigration_Module.xlsx

For immigration, the ONS of Mauritania currently uses DemProj without emigration and no “ready-made” official emigration data are currently available. This gap is closed by using a 12 percent sample of the 2013 census to model emigration.

Emigration rates by five-year age groups were calculated from 2012 departures from households reported in the 2013 census. Following this approach, only emigrants who leave part of the household are covered.

Due to sample size restrictions and other data quality issues, we currently parameterize the model on the national level only. In the available sample, 88 percent of emigrants are male. Calculating survival of staying in the country from these period observations and ignoring back migration assumes that 50 percent of males would have left the country at age 60. Similar to internal migration and immigration, the model allows the option to switch emigration off.

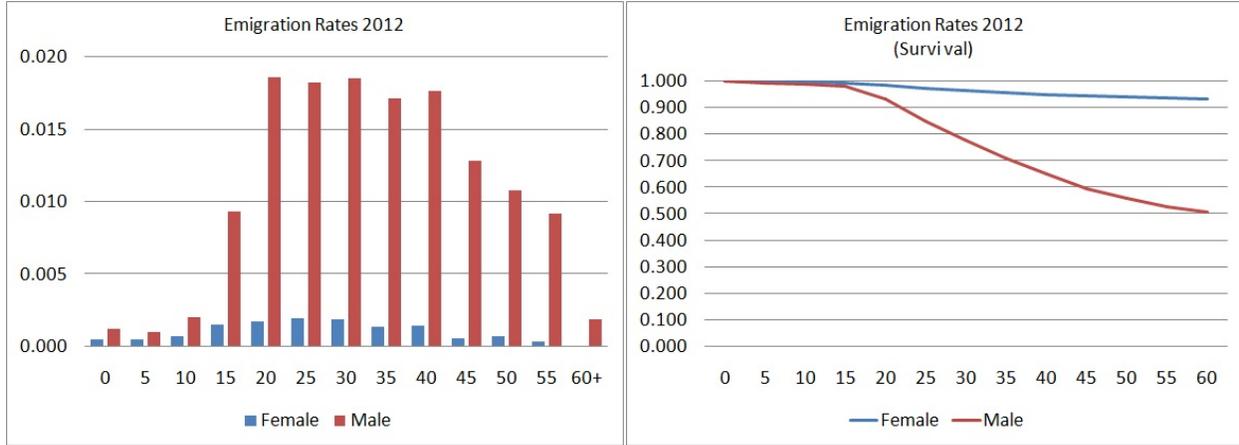


Figure 2-9: 2012 Emigration Rates

The parameters are based on two simple tables calculated from the census file version, including recent emigrants.

```

/* ***** */
/* DATA: POPULATION INCL. EMIGRANTS */
/* ***** */

clear all
#delimit;
use "Microdata\m_census_all_incl_emigrants.dta";

/* ***** */
/* VARIABLES */
/* ***** */

/* Age 12 months ago, drop if not born then */
gen m_age2012 = int(M_AGE)-1; drop if m_age2012 < 0;

/* 5 year age groups */
gen m_agegr = int(m_age2012 / 5) * 5;

/* identify emigrants */
gen m_emigrate = 0; replace m_emigrate = 1 if M_POR==13;

/* ***** */
/* ANALYSIS */
/* ***** */

tab m_agegr m_emigrate [iweight = M_WEIGHT] if M_MALE==0;
tab m_agegr m_emigrate [iweight = M_WEIGHT] if M_MALE==1;

```

Chapter 3. Phase 2: Extending the Model Showcasing the Power of Micro-simulation

In the second phase of model development, we extend typical population projections in the following ways for education, first union, fertility, provinces, and infant mortality:

- **Education:** Education is modeled and used for other processes as an explanatory variable. Currently the education module concentrates on primary education, central to current policy efforts in Mauritania, which does not provide universal access to primary education and has low retention rates. The model includes entry into primary and graduation from primary as key processes, extended by a distributional model of years of primary completed by school dropouts. The model works on the provincial level and can account for inter-generational transmission processes, i.e., children from higher-educated mothers having higher chances of entering and graduating from school. While useful for some policy applications in Mauritania, the model also provides a platform for refinement, e.g., by adding higher graduations.
- **First union:** The time of first union formation—which in Mauritania is marriage—is contained in many censuses in developing countries and is an important deterrent of fertility. It allows a better depiction of the concentration of reproduction: instead of distributing children to women independent of union status (especially in early life, e.g., 20 percent of females without schooling are married at or before age 14 in Mauritania), fertility is concentrated to fewer women, thus better reflecting reality. Change to union formation is one of the key mechanisms of fertility changes, as the age at first union increases fast especially for those entering higher education. This allows for addressing fertility change to various factors.
- **Fertility:** Fertility is modeled in detail and allows the user to choose a wide range of scenarios for alternative model implementations. The goal of detailed modeling is a realistic depiction of the distribution of family size. This requires modeling fertility by parity, time since last birth, as well as characteristics such as partnership status and education, a key determinant. In addition, the fertility module allows alternative alignment routines, reproducing given projections in aggregated outcomes, while adding realistic family sizes. Distribution of family size—the concentration of reproduction—may be of key policy relevance, as it is frequently those with the least education who have more children. Parity and education are also key explanatory variables for maternal and child mortality.
- **Provinces:** Mauritania has 13 provinces (i.e., Willayas) and most processes are modeled on the provincial level. The biggest Willaya is the capital, Nouakchott, with one-third of the national population; it also receives the most migrants—approximately 75 percent. Inter-provincial migration is modeled in alternative ways, with or without accounting for education, which considerably increases mobility in the Mauritanian case. As for other modules, the migration module can be a platform for model refinement.
- **Infant mortality:** Infant mortality is modeled in some detail, using a list of explanatory variables including age and education of the mother. This allows for modeling downstream effects of educational investments and other potential policies to reduce infant mortality.

The following general design choices were made:

- We kept the model simple, transparent, and user friendly, selecting models that allow intuitive parameters. For example, we use probabilities, hazard rates, and odds ratios whenever possible, avoiding complex regression coefficients.
- We covered a variety of approaches typically used in micro-simulation, including hazard regression, logistic regression, life tables, and, in the case of first union formation, a parametric model which allows very intuitive parameterization, e.g., by setting average ages at union formation and the proportion of women expected to enter marriage over their lives, rather than measures that would require math to calculate such properties.
- We avoided conflicts between goals like simplicity, variety, and use of established models by implementing various alternative solutions in parallel when such conflicts arise, letting the user decide which approach to use. This will also help demonstrate and assess the impacts of alternative assumptions on the simulation results.
- We built and refined the model step-wise, and allowed switching some of the added processes on or off in the simulation, which offers a broad range of scenarios. For example, the model can be run with and without internal migration, and when adding education as a variable influencing migration risks, we can compare results, such as the size and education composition of sub-national populations, with simulations that do not account for education.
- Starting from an established macro model, we reproduced published projections. This can be done directly, as in step one, by using the same assumptions and reproducing the same processes, or indirectly, by aligning aggregated outcomes of the model to these targets, while adding detail and additional distributional information.
- We built the model as a platform that accepts future extensions, refinements, and modifications. The technical implementation is modular, allowing model builders to re-use code.

3.1. Primary Education

3.1.1. Discussion

In its current version, the education module focuses entirely on primary education. Three education levels are available: never entered primary school, entered primary school without completion, and completed primary school. In this way, modeling the decision on education outcome can be accomplished early in life, before other processes, such as union formation and first birth, for which education is used as an independent variable.

The model is driven by three probabilities: to graduate from primary education by year of birth, province of birth, and sex, which are entered respectively. As another dimension and model option, mother's education can be added as a relative factor. When this option is chosen for a selected year of birth onward, the probabilities to enter and graduate primary education by mother's education are "frozen" and all future changes are driven by the changing educational composition of mothers. This allows analysis of trends, as the relative differences by parents' education are typically very persistent.

Modeling of primary education is only the start of modeling education; higher education may be the development priority of the model and primary education may serve as template. While the model is very simple, we implemented it in a way that allows extensions and refinements in many dimensions. For example, we implemented a separate primary school actor who selects students. This approach can be easily extended to model both the demand and supply side of schools and/or the demand for teachers on a regional level.

3.1.2. Parameters

The primary education module contains nine parameters:

- The probability to enter primary education by year of birth, sex, and province of birth
- The probability to graduate from primary education by year of birth, sex, and province of birth
- Primary school entry age
- Primary school graduation age
- Start of the school year
- Use of mothers' education: on/off
- Log odds of entering primary education by mother's education
- Log odds of graduating from primary education by mother's education
- Year from which on mother's education is introduced and for which the model is calibrated

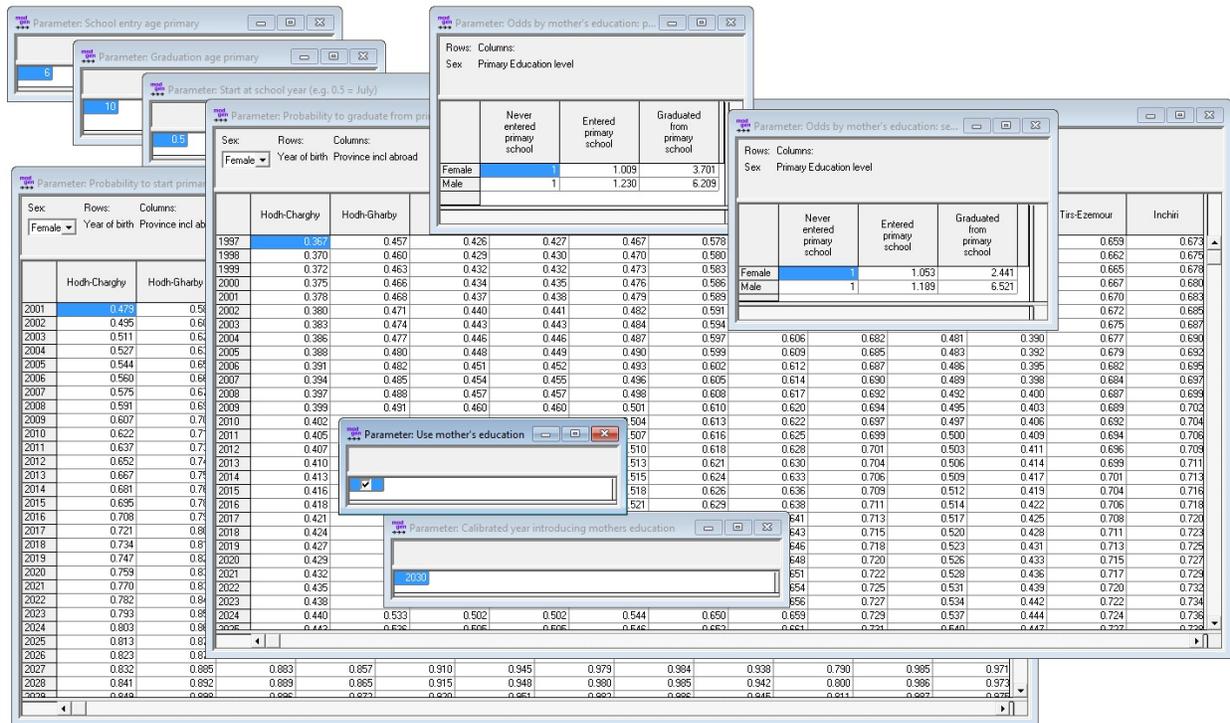


Figure 3-1: Education Parameters

3.1.3. Analysis

The core models

The primary education module is based on four regression models. The first two—entering primary education and graduating from primary education—are based on census data and are used for the base scenario. In the second step, related models are estimated from MICS to estimate the influence of mother's education, and mother's education can then be added to the model as a proportional factor, allowing for a variety of scenarios. The key parameter tables for entering and graduating from primary school contain probabilities which were calculated from log odds estimated by logistic regression:

$$P = \exp(\text{estimated log odds}) / (1 + \exp(\text{estimated log odds}))$$

The base models are estimated from census data and use birth, sex, and year of birth as variables. Cohort trends are estimated using people between age 12 for primary school and 16 for secondary, and age 32. These trends are a good fit for the past and, due to the log transformation in logistic models, level off in the future, providing plausible base scenarios. Differences by province are very pronounced and persistent.

In the second step, odds ratios of primary and secondary school graduations by mother's education are estimated from MICS. Users can run alternative scenarios in which overall trends in education progressions are replaced by the dynamics stemming from mother's education. The application automatically calibrates results for a chosen base year of birth, so that the alternative scenario, including mother's education, produces the same outcome as the trend scenario for the chosen year of birth.

As the model runs on a provincial level, education trends stemming from three sources can be distinguished:

1. The changing composition of the population by province
2. Changes in education stemming from the changing education composition of the mother's generation
3. Overall cohort trends

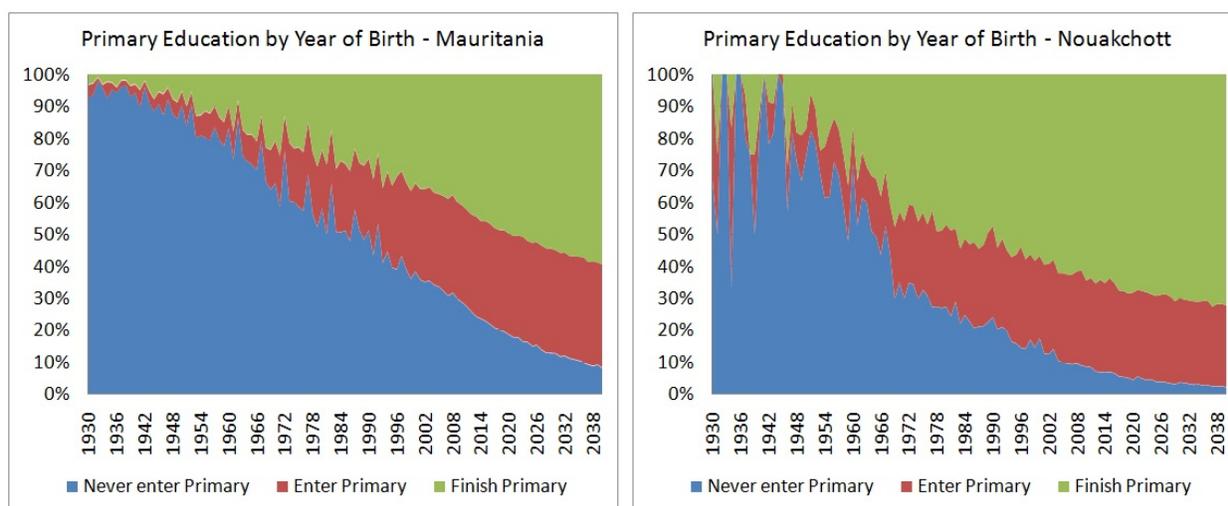


Figure 3-2: Highest School Attended, National and Nouakchott - Data and Projected Trends by Year of Birth

To detect gender-specific trends, separate logistic regressions by sex were estimated, including the population age 12-32 and dummies for province (at birth), assuming proportionality. Age 12 is used as the age cut-off, assuming that at this age everyone entering primary education has done so. Data show the lowest proportion of people without education at this age. There are considerable differences by province; the percentage of 12-year-olds who never entered formal schooling is between 15 percent in Nouakchott and 55 percent in Hodh Charghy. These two provinces also have the highest populations. Alternative scenarios could be created in which these gaps are closed. There are also gender differences; females have a steeper trend and recently overtook the males.

The model for primary school graduation follows the same logic, and the only difference is that age 16 was selected as the cut-off age, with the highest rate of people who have graduated from primary. Retention rates, which are the percentage of people staying in primary until graduation, are low, between 40 percent in Hodh Charghy and 70 percent in Nouakchott. Again, female trends are steeper, but male rates are still higher, especially in provinces with low retention rates.

	Enter Primary		Complete Primary	
	Female	Male	Female	Male
Hodh Charghy (reference)	0.0000	0.0000	0.0000	0.0000
Hodh Gharby	0.4431	0.1316	0.3724	0.3461
Assaba	0.4193	0.4426	0.2466	0.1664
Gorgol	0.1908	0.4244	0.2491	0.1756
Brakna	0.7113	0.7871	0.4137	0.3777
Trarza	1.2434	1.2649	0.8570	0.7761
Adrar	2.2445	2.1620	0.8967	0.7951
Dakhlett Nouadibou	2.5432	2.5471	1.2290	0.8672
Tagant	1.1158	1.2529	0.3871	0.3404
Guidimagha	-0.2757	0.0959	0.0159	0.2851
Tirs-ezamour	2.5892	2.6865	1.2050	1.1571
Inchiri	1.9248	2.0575	1.2644	0.8602
Nouakchott	2.1536	2.0623	1.2340	0.9357
Abroad	0.0960	0.1576	0.7278	0.5350
Age in years (12-32 resp. 16-32)	-0.0648	-0.0381	-0.0113	-0.0026
Constant	0.6280	0.1525	-0.3747	-0.1228

Table 3-1: Regression Results for Entry and Completion of Primary Education

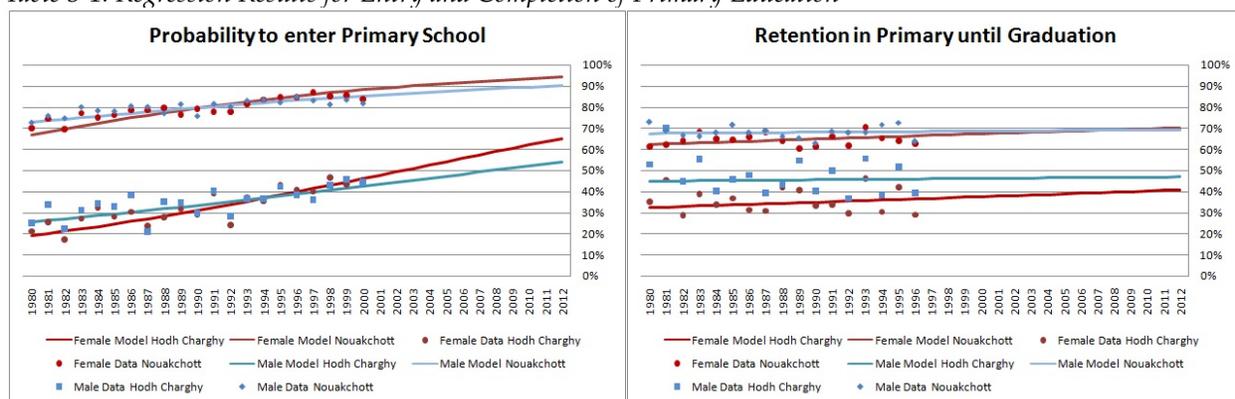


Figure 3-3: Primary Education by Year of Birth for Nouakchott and Hodh Charghy, Data and Projected Trends

Introducing Mother's education

Models including mother's education cannot be estimated from the census, so MICS was used to run the same models as above to control for differences by Willaya and extend by mother's education. The estimated values are in line with literature; interestingly, mother's education has a far higher impact for males.

	Enter Primary		Complete Primary	
	Female	Male	Female	Male
Mother never entered primary (reference)	0.0000	0.0000	0.0000	0.0000
Mother entered Primary	0.0094	0.2067	0.0521	0.1734
Mother entered Secondary	1.3085	1.8260	0.8923	1.8750
Hodh Charghy (reference)	0.0000	0.0000	0.0000	0.0000
Hodh Gharby	-0.5918	-0.3268	0.0779	0.0447
Assaba	-0.3327	-0.2350	-0.4629	0.3378
Gorgol	-0.3672	-0.0627	0.3991	0.4301
Brakna	0.2819	0.3511	-0.1856	0.6851
Trarza	0.1730	0.2885	-0.2307	0.9342
Adrar	1.7310	1.6031	1.3301	0.5411
Dakhlett Nouadibou	1.0285	1.6523	0.7354	0.6433
Tagant	0.7785	1.0559	-0.0157	0.6493
Guidimagha	-0.2053	0.1389	-0.3497	0.4121
Tirs-ezemour	1.4868	2.9264	1.3307	1.8395
Nouakchott	0.5371	1.0290	0.7083	1.0210
Age	-0.0269	-0.0212	-0.1264	0.0819
Constant	1.6904	1.0809	1.7987	-1.8424
Odds: exp(coefficient)				
Mother entered Primary	1.0094	1.2296	1.0535	1.1893
Mother entered Secondary	3.7006	6.2088	2.4407	6.5209

Table 3-2: Estimation Results for Odds by Mother's Education (Data: MICS 2011)

Combining the census model with estimates for odds by mother's education requires calibration to get the same probabilities for a given base year with and without introducing mother's education. Technically, one searches for a calibration term (with the interpretation of an additional proportional log odds factor) by numeric simulation, which, for a given education composition of the mother's generation, leads to the same graduation probability as the model without mother's education. These calculations are done automatically within the model software.

Reproducing the model

- Stata code: 07_Education_MICS.do
- Excel parameter file and census analysis results: Education_Module.xlsx
- For analysis and regressions based on MICS data, run 07_Education_MICS.do. Result tables can be copied and pasted into the sheet MICS. This updates the graph of education by age and updates four regression models (enter and finish primary education by sex) used for calculating the parameters of odds by mother's education.
- For analysis and regressions based on the census, run 06_Education_CENSUS.do and copy and paste result tables into the sheet CENSUS. Tables are education by age, regression results for entry, and success in primary education by sex (four models). There are eight other output tables used to produce graphical examples for two Willayas, Hodh Charghy and Nouakchott.
- All model parameters are calculated automatically in the sheet PARAMETERS, from which parameters can be copied and pasted directly into the application.

- The sheet SIMULATIONS can graphically compare the results of three alternative simulation results. Result tables from the application can be copied and pasted into this sheet to update all graphs.

3.2. Union Formation

3.2.1. Discussion

Changes in the age of first union formation is one of the key mechanisms behind fertility changes. Many developing societies currently experience a rapid increase in that age, partly resulting from educational expansion. In Mauritania and many developing countries, first union formation on average still occurs very early in life. Between 10 and 40 percent of females, depending on education, have entered a union by age 17; about 20 percent of women without any formal schooling enter a first union at or before age 14. Including this variable in the fertility model allows a better depiction of the concentration of reproduction: instead of distributing children to women independent of union status, fertility will be more concentrated to fewer women, especially at young ages, which better reflects reality.

From a modeling perspective, the fast-changing age profile in first union formation imposes interesting challenges. Fast demographic change, especially shifts in age profiles, make period data of limited use for modeling. For example, union formation rates decrease fast for the very young, but this observation does not necessarily mean that fewer people enter a union over the life-course, even if union formation rates are currently low at higher ages, where most of people observed today have entered a union already. It is in this environment when parametric models demonstrate their power. In our case, we implemented the Coale & McNeil approach for modeling entry into first unions. The parameterization of such a model is very intuitive; parameters are the minimum age at first union formation (currently around 10 in Mauritanian data), the average age, and the proportion of women who will eventually marry. We allow separate parameterization by primary education outcome. Period rates that result from fitting relevant models using these parameters are calculated within the application and displayed in output tables. Alternatively, to give users a broader choice in approaches, we allow period rates by education as model parameters. Also, the The Coale & McNeil model produces table output of union formation by age, which can provide feedback as a parameter for the second model. This can easily create scenarios of policies that set a minimum age at marriage.

3.2.2. Parameters

The module has three parameters:

- A model selection parameter choice between a Coale & McNeil model and a model based on age-specific hazard rates
- The parameter for the Coale & McNeil model: minimum and average age at first union formation, proportion of women ever entering a union by year of birth
- Union formation rates by year of birth and age

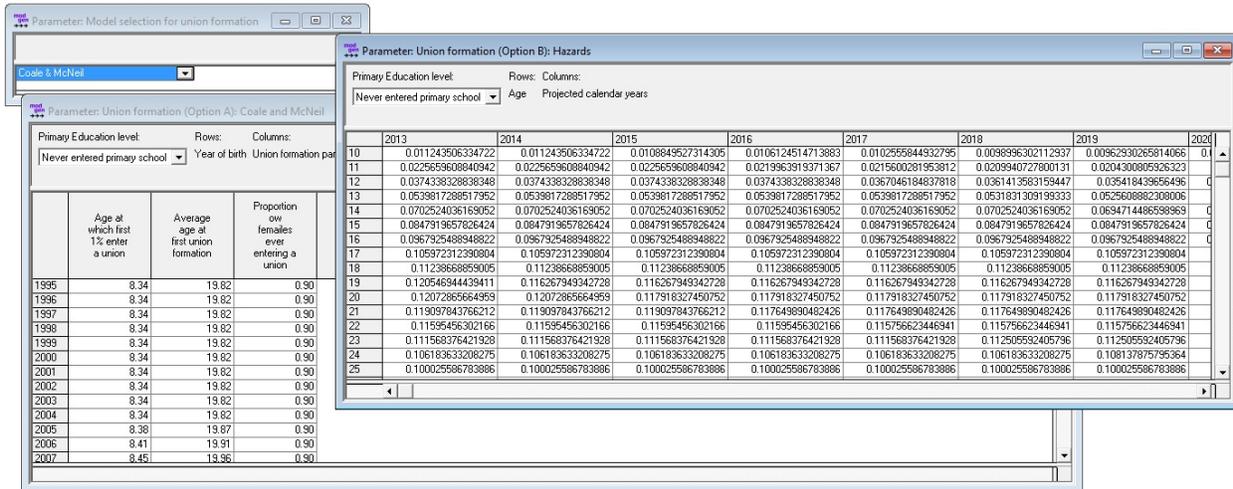


Figure 3-4: Union Formation Parameters

3.2.3. Analysis

Current period data

From a modeling perspective, the fast-changing age profile in first union formation imposes interesting challenges. Fast demographic change, especially shifts in age profiles, limit use of period data for modeling. When applying age-specific period data of first union formation as observed in the census, between 20 and 25 percent of women, depending on education, would never enter a union before age 40..

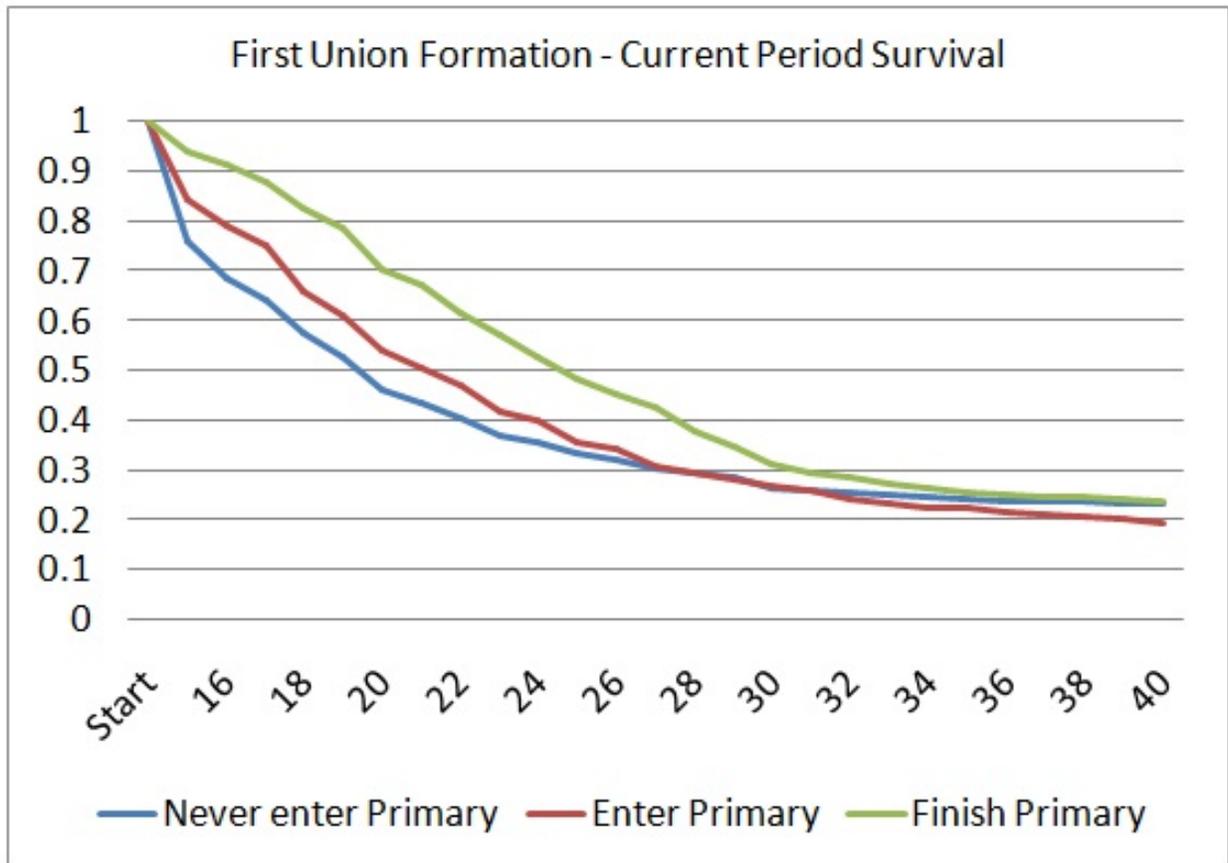


Figure 3-5: Current Period Survival

Hazard regression, including age-specific trends

Comparable results for period rates can be obtained by hazard regression, including age-specific trends. The following graph shows the very distinct age pattern in the most recent union formation hazards by education for observations between 2005 and the 2013 census.

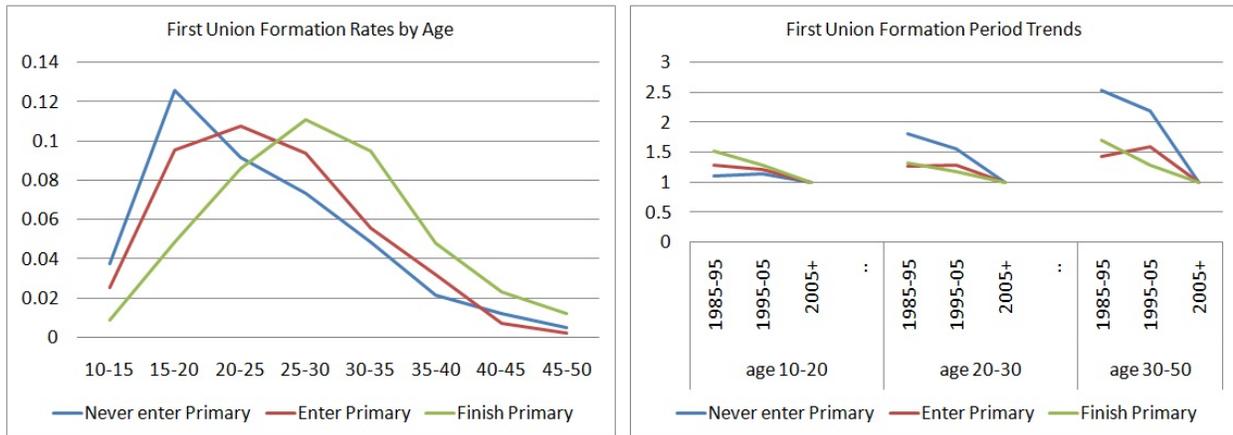


Figure 3-6: First union formation rates by age and time trends

Based on these rates, and creating a synthetic cohort living its entire life in the period 2005-13, almost 90 percent of women would eventually enter a first union. There are very steep trends. Relative period risks (reference 2005+) are decreasing for all distinguished age groups, but in reverse order by education and age group: rates below 20 were decreasing most significantly for more highly educated women, while for older age groups, rates decreased more for groups with less education.

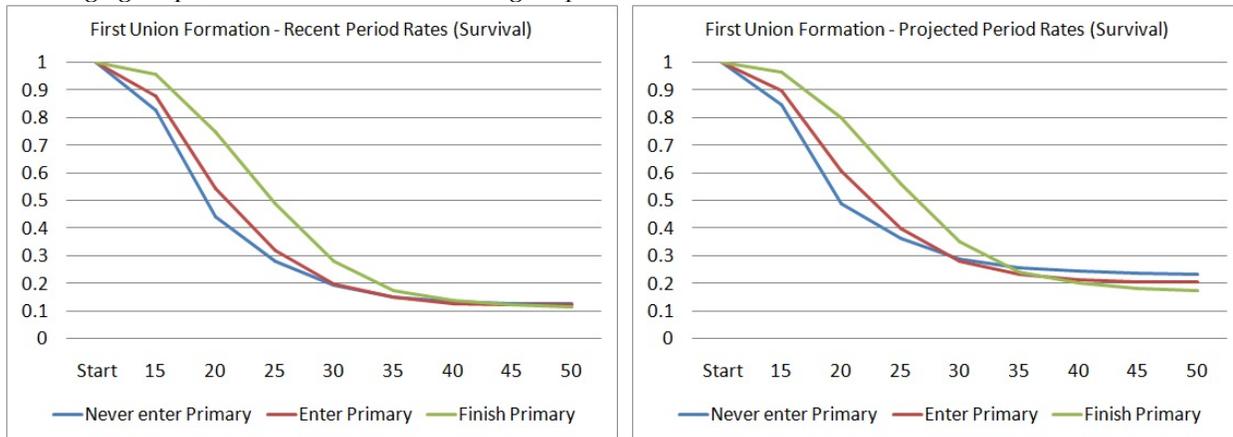


Figure 3-7: Recent Period Survival and Projected Survival 2015-2025

Past period trends are of limited use for informing future scenarios. It is expected that delays in union formation at lower ages will eventually lead to catch-up effects at higher ages, but this reversal in trends was not observed yet, using the broad age groups in this example. While “recent trend scenarios” are popular in micro-simulation, continuing the trend as observed in the past 10 years for another 10 years would double the proportion of women never entering a union.

Hazard regression by cohort

Displaying rates by birth cohorts makes the modeler’s choices visually more explicit: the younger the cohort, the shorter their observed behavior and the longer the remaining lifetime for which rates have to be modeled.

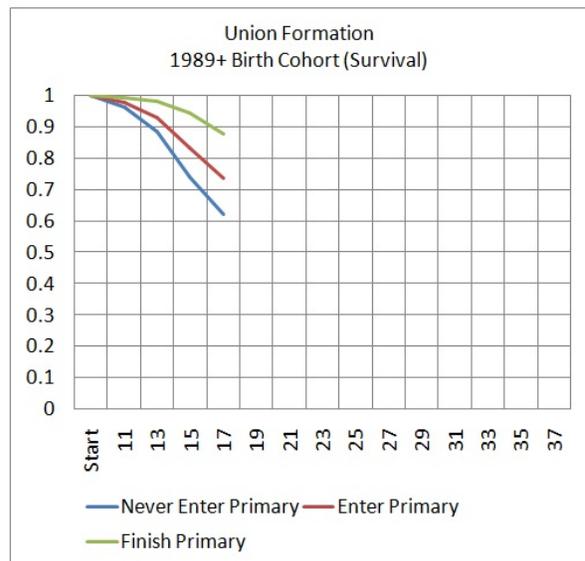
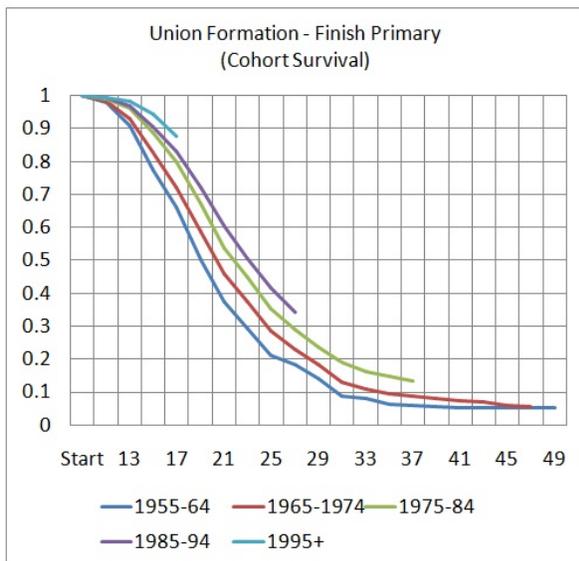
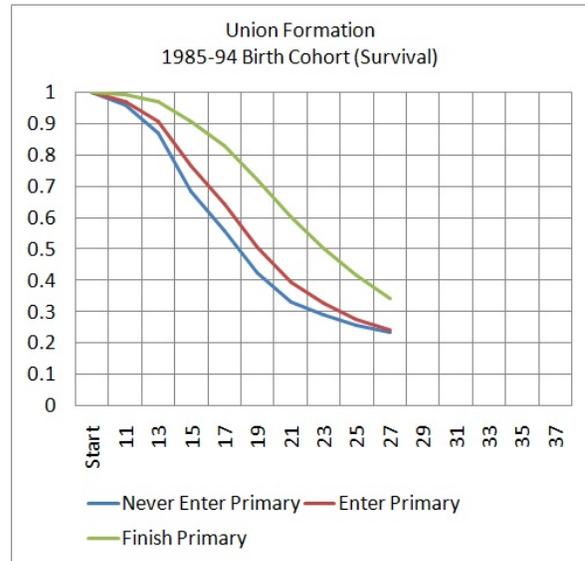
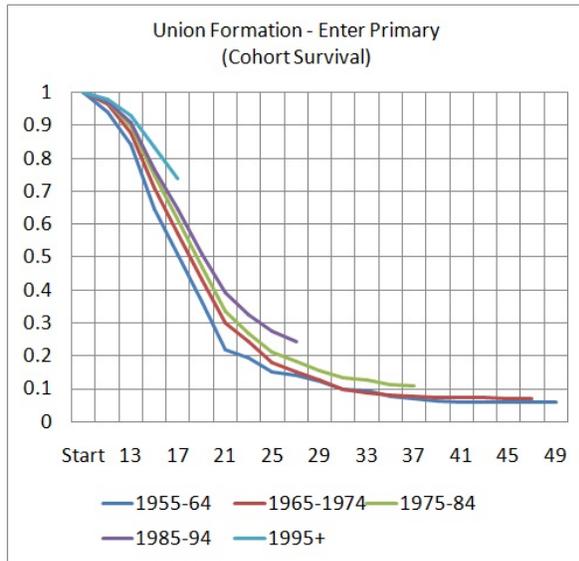
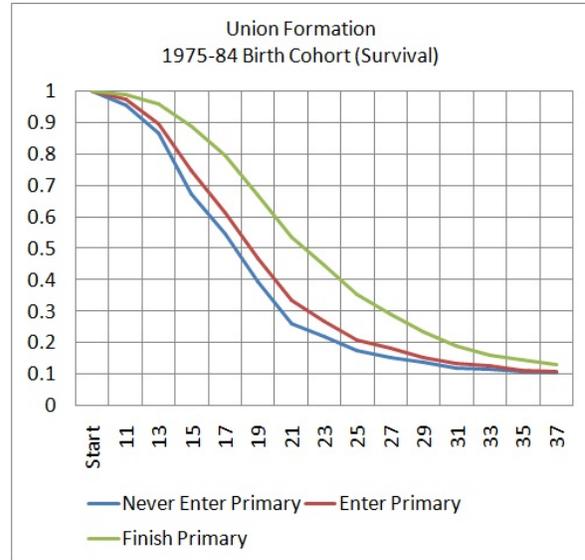
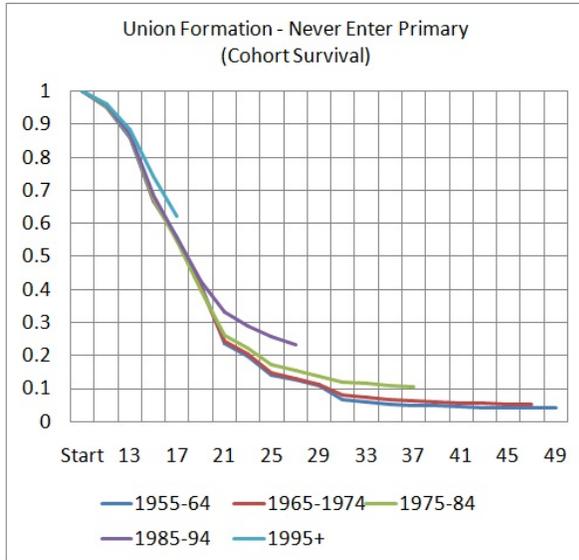


Figure 3-8: Cohort Survival

Parametric models

Besides “borrowing” from observed behaviors of older cohorts with or without applying trends, another approaches are parametric models to fit survival curves. Such a model was proposed by Coale and McNeil in the 1970s, based on extensive studies on the age pattern of first marriages in many countries. Their standard density of first marriage, constructed from pattern observed in Sweden 1856-9, has the form:

$$g_s(x) = 0.19465 * \exp(-0.174(x-6.06)) - \exp(-0.288(x - 6.06))$$

They found that a relational model with three parameters transforming the proportion of ever married at the age resulting from the above density function can fit most populations:

$$G(a) = C * GS ((a-a_0) / k)$$

- a = age
- a0 = minimum age when the process starts (empirically, ~age where the first percent married)
- k = indicator of the spread of the distribution, i.e., how fast marriage occurs after a0 (how many years of the population’s schedule are equivalent to one year of the standard)
- C = the proportion of the population eventually entering marriage

Rodriguez and Trussell (1980) noted that k stands in direct relation to the average age at marriage μ , allowing parameterizing a model more intuitively with μ instead of k:

$$\mu = \int aG(a)da = a_0 + 11.36k, \text{ thus } k = (\mu - a_0) / 11.36$$

Estimations for Mauritania: using the Excel Solver to fit the equations based on the survival curves provides remarkably good fits to data. Figure 3-9 shows fitted curves for women with secondary education as well as the cohort trends in average age at first union formation:

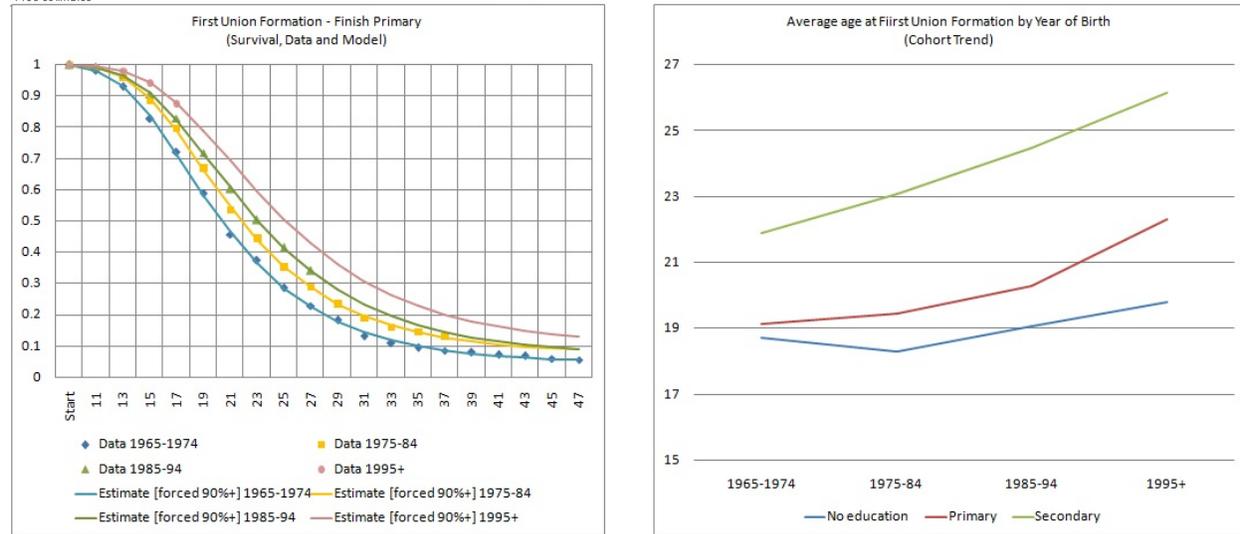


Figure 3-9: Solution for Coale and McNeil Model, Time Trends

The highest uncertainty (and room for scenario building) is probably in “guesses” for C. Almost equally good fits, especially for the youngest cohort, can be found fixing C at different levels.

In the context of the micro-simulation, this approach was implemented as a choice, side by side with a

table of period rates, because it is probably more intuitive for scenario building. It allows for parameterization—by birth cohort, separately for each of the three education groups—by setting parameters for average age at union formation and the proportion of women ever entering a union. Scenarios for parameters for average age can be informed by recent trends or by international experience.

Reproducing the module

Stata Files:

- For analysis of current period probabilities, run 08_UnionFormation_Prob.do and copy the model results (three models by education) into the sheet PeriodProbabilities. This is for analysis only and not used in the application.
- For analysis of recent period rates and period trends, run 09_UnionFormation_Histories.do and copy the results, three hazard regression models by education, into the sheet PeriodHazard&Trends. This is for analysis only and not used in the application.
- For analysis of cohort survival, run the file 10_UnionFormation_Cohorts.do and copy and paste the estimation results, 15 hazard regression models by cohort and education, into the sheet Cohort Survival. These results are used for curve fitting to fit Coale and McNeil models. Curve fitting is performed using the Excel Solver to find solutions for three parameters: Start age, Average age at marriage, and Proportion eventually marrying. Solutions must be found for four cohorts for each of the three education groups. This is done in the three sheets (one by education): CohortNeverEnter-Primary, CohortEnterPrimary, CohortFinishPrimary. All three parameters can be searched for simultaneously, and parameters can be fixed, e.g., to force the model to produce a chosen target proportion of people ever marrying.

Excel

- All calculations and parameter are in the workbook: Union_Formation.xlsx
- The model Parameters are provided in the sheet PARAMETERS from where they can be copied and pasted into the application. This sheet also allows setting future target parameters (for 2050) and applies linear trends to reach them.

3.3. Alternative Models for Fertility

3.3.1. Discussion

While very popular, the use of period rates of age-specific fertility in population projections has three types of limitations:

- Even if it were possible to have perfect knowledge of future rates, the simulation would not produce realistic life-courses for women. Age as the only determinant of fertility would not lead to a realistic distribution of children to mothers and thus of family sizes. It ignores other determinants of fertility like parity, time of the last birth, union status, and education. So even with perfect foresight, we would obtain only the right number of babies. Micro-simulation can increase policy relevance of projections by placing newborns into their family context: their survival, future school attendance, etc., can take into account parents' characteristics, which in return are relevant to the cost of policies.
- A projection tool based on age-specific rates cannot inform scenarios of fertility changes. Scenarios must be produced outside of the model, and often mechanically, by starting from observed rates, future target rates (e.g., from a country which underwent demographic changes earlier), and a transition path. Theories of behavioral changes are not made explicit nor are they distinguishable from composition changes. Micro-simulation allows the explicit modeling of behavioral changes, respectively, permits the study of composition effects in isolation. The latter can run status quo scenarios

where changes in overall fertility arise entirely from changing composition of the population.

- Macro scenarios based on given rates do not allow modeling of feedback reactions and policy effects on fertility. For example, if highly educated women are expected to have fewer and later births, a micro-simulation model will naturally produce these downstream effects when running scenarios on educational expansion. Other scenarios could include legislation effects, e.g., the imposition of a minimum age of marriage.

To build and use the strengths of micro-simulation, we can derive the following modeling priorities:

- The capability to produce realistic life-courses for given overall rates. In this mode of model use, the focus is on the refinement of a macro model—a realistic distribution of children to mothers—without changing the macro assumptions on the future number of births.
- The capability to model and analyze scenarios of behavioral changes as well as composition effects.
- Scenario support for the modeling and identification/assessment of downstream effects of policies, like educational expansion or legislation of legal ages for marriage.

The refined fertility module was built around these priorities. First, we model births separately by birth order. In the case of first birth, age is kept as the main (baseline) factor, but age-specific birth rates are estimated separately by education group (expecting differences in the age pattern) and for women who ever entered a union and those did not (where fertility is very low).

For higher-order births, we estimate separate models for births of order 2-15. To model realistic birth intervals, the baseline is now the time since previous birth. Relative risks were added to capture the effects of education and broad age groups, for higher ages where fertility decreases considerably.

Users are given various model choices for model selection and alignment options. In particular, the refined model can be run with total births or total births by age aligned with the base model. This produces identical aggregate projections as found in published sources while respecting the relative differences in birth risks between women by parity, birth interval, education, and union status.

3.3.2. Parameters

The refined fertility module introduces four new parameters:

- Model selection: Users have four choices for modeling fertility
 - Option 1: Using the base model; the refined module is disabled
 - Option 2: Using the refined model without alignment
 - Option 3: Using the refined model but aligning the aggregated number of births to the base models. While the relative fertility differences between women of different parity, education, union status and age are respected, the model is aligned (forced) to produce the same number of births as the base model. This produces the same aggregate scenarios, but adds detail leading to more realistic female life-courses.
 - Option 4: Using the refined model, but aligning the aggregated number of births by age to the base models. While the relative fertility differences between women of different parity, education, and union status are respected, the model is aligned (forced) to produce the same number of births by age as the base model. This produces the same aggregate scenarios, but adds detail leading to more realistic female life-courses and is faster than Option 3, as at each birth event, only the most appropriate mother is searched within the age group.
- First birth rates by age, province, education, and union status
- Higher-order births: duration baseline and relative risks by mother's age and education for births of order 2-15

- Period trends by parity

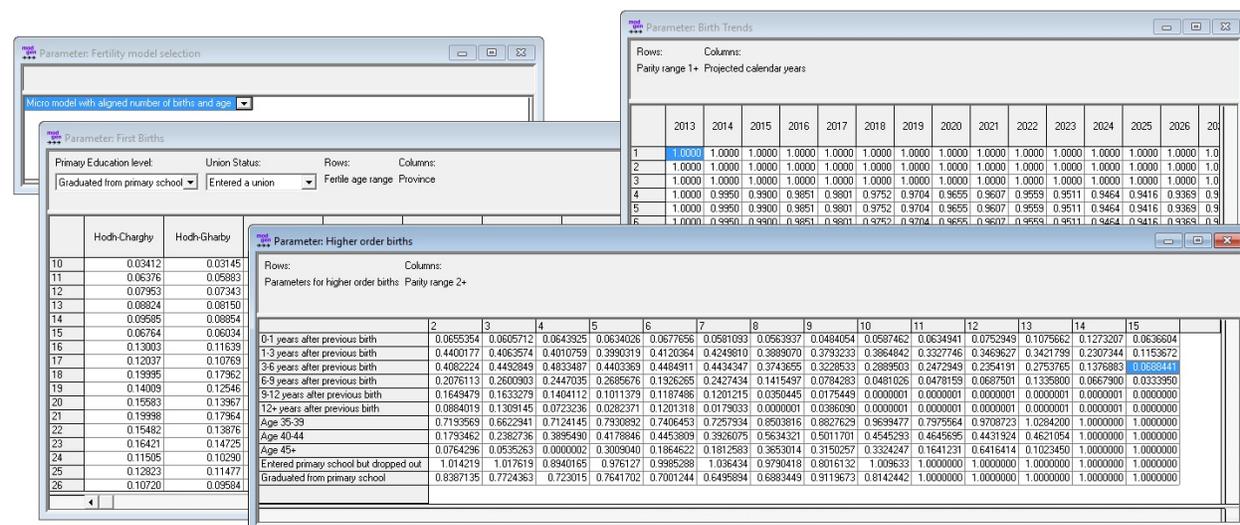


Figure 3-10: Fertility Parameters - Refined Version

3.3.3. Analysis

First births

First births are based on estimates from census data using the information on reported births in the past 12 months. Parameters are birth rates (hazards) by single year of age and province of residence, marital status (ever having entered a union), and the three education groups based on primary school entry and graduation. While the parameter consists of a single four-dimensional table, i.e., age, province, marital status, education, the respective rates are calculated from the log odds and estimated using logistic regression models:

- For women 15+ who ever entered a marriage, models by age and province are run separately for each education group, allowing for different age profiles by education.
- For women 15+ who never entered a marriage, due to a much smaller population and very low birth rates, education is added as proportional factor instead of estimating the model separately by education.
- Fertility below age 15 is estimated separately, as the influence of education and regional differences are different at this young age compared to women 15+.
- The models for first births of women below age 15 are estimated separately by marital status. Due to very small sample/population, inter-provincial differences are not estimated for women who never entered a marriage.

Higher-order births

For the second to the 14th birth, separate proportional hazard regression models are estimated from MICS data. This approach was chosen as the duration since last birth is a strong predictor of higher-order birth events and including this duration, which is not available from the census, allows more realistic modeling of birth intervals and thus female life-courses. Also, using event history data informs scenarios of future trends.

Reproducing the analysis

- Excel file of parameter tables and calculations: Refined_Fertility.xlsx

- Stata file for first births (based on census): 12_FirstBirthCensus.do
- Stata file for higher-order births (based on MICS): 13_HigherOrderBirthsMics.do
- Stata file: 11_MICS_CreateAnalysisFile.do creates the analysis file for higher-order births based on a file of women and a file of children. The file has the following variables:

```
File: m_mics_fertility.dta
```

```
Variables:
```

```

- M_ID      Person ID (women)
- M_BIRTH   Birth date (women)
- M_WEIGHT  Record Weight
- M_Bnn     Birth dates of children nn 01-14
- M_EDUC    Primary education: never entered / dropout / finish level 5+
- M_MAR     Date of first marriage
```

3.4. Infant Mortality

3.4.1. Discussion

Child mortality is a key policy concern in many developing countries. There is a wide body of literature on determinants, both individually, e.g., characteristics of the mother, and contextually, e.g., availability of health care infrastructure on the regional level. Micro-simulation can be a powerful tool for policy-relevant analysis and projections, as it can handle the detailed characteristics that drive child mortality. In this context, the presented model is just a starting point for detailed analysis. It takes two characteristics of mothers into account, mother's age and education. In combination, we found child mortality 2.5 times higher for mothers who did not graduate from primary school and are under 15 years old.

The child mortality module is optional. If activated, it starts replacing the overall mortality model five calendar years after the starting year of the simulation, ensuring that all children "know" their mother's characteristics (are born in the simulation). As mother's characteristics are not available for children born outside the country, immigrants are excluded from the model and handled by the general mortality module.

3.4.2. Parameters

Four parameters were added to the base mortality module to model child mortality at a greater level of detail:

- A model selection parameter, with four model choices
 - Disable the child mortality model: the same model as for all ages is used
 - Child mortality model without alignment: the child mortality module replaces the overall mortality module for ages 0-4. Note that the overall life expectancy might be altered by this choice.
 - Child mortality model calibrated for an initial year, then trends for other ages: in this choice, life expectancy is the same as in the overall mortality for the given year, but as the composition of the population by mothers' characteristics changes over time, the number of deaths and therefore life expectancy will differ for the following years, allowing for scenario comparisons.
 - Child mortality model calibrated for an initial year, then specific trends from the child mortality module: Here, life expectancy is the same as in the overall mortality for the given year, but as the composition of the population by mothers' characteristics changes over time,

and as a result of potentially different trends, the number of deaths, and therefore life expectancy, will differ for the following years, allowing for scenario comparisons.

- Baseline risks for child mortality (reference category mothers above 16 and graduated from primary school)
- Relative risks for child mortality by child’s age, mother’s age (group) at birth and mother’s education
- Time trend for child mortality by age

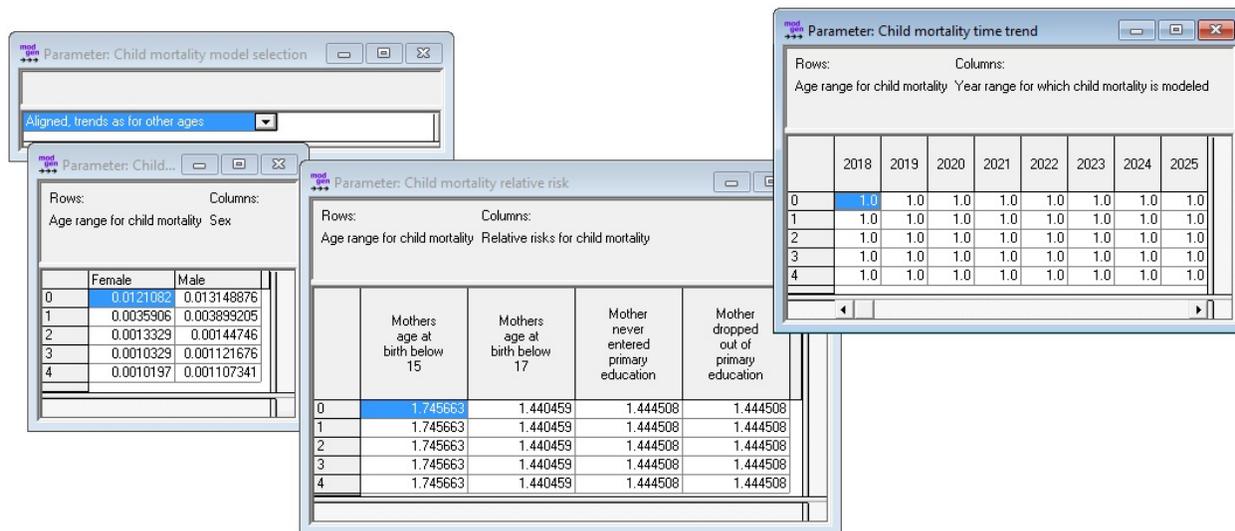


Figure 3-11: Child Mortality Parameters

3.4.3. Analysis

Data analysis is based on the file **m_mics_childmortality.dta** generated from MICS and containing the following variables:

```
m_mics_childmortality.dta

- M_BIRTH      Date of birth (months since 1900)
- M_DEATH      Date of death (months since 1900)
- M_MALE       Sex (Male = 1 / Female = 0)
- M_WEIGHT     Weight
- M_AGEMO     Age of mother at birth (in months)
- M_EDUCMO     Primary education of mother (0 never entered, 1 dropout, 2
graduate)
- M_INTERV     Date of interview (months since 1900)
```

- Stata file: 16_MICS_AnalysisChildMortality.do
- Parameter calculation and tables Excel file: ChildMortality_Module.xlsx

Baseline hazard and relative risks were estimated simultaneously by a proportional hazard regression model. There were not any significant difference between not having entered primary school at all and having dropped out from primary school, so we collapsed the two groups.

```
/*
ANALYSIS OF CHILD MORTALITY
DATA FROM MICS MAURITANIA
MARTIN SPIELAUER, 2016
```

```

Input file: m_mics_childmortality.dta
*/
/*
-----
Wrapper function stpiece
-----
the following adds the wrapper function stpiece which can be used to make code more
efficient;
without this function, time splits of the baseline duration have to be done
manually;
it was developed by Jesper B. Sorensen Nov. 20 1999;
the function can be added to Stata, download at:
http://fmwww.bc.edu/repec/bocode/s/stpiece.ado;
*/

clear all
program define stpiece
version 6.0
syntax [varlist(default=none numeric)] [if] [in] , ///
      [tp(numlist >=0) tv(varlist) PRESplit(integer -1) noPREServe *]
if "`preserv'"==" " {
  preserve
}
if "`presplit'" == "-1" {
  if "`tp'"==" " {
    display in red "You must either declare data to be presplit or use tp()"
    exit 198
  }
  display in gr "Invoking stsplitsplit..."
  quietly stsplitsplit tp, at(`tp')
  display in gr "Creating time pieces..."
  quietly tab tp, gen(tp)
  local i = _result(2)
  drop tp
}
else {
  local i = "`presplit'"
}
if "`tv'" ~= "" {
  display in gr "Creating interaction effects..."
  tokenize `tv'
  while "`1'" ~= "" {
    local stub = substr("`1'",1,4)
    local j=0
    while "`j'" < "`i'" {
      local j = `j'+1
      local x = "tp`stub'`j'"
      quietly generate `x'=`1'*tp`j'
    }
    mac shift
  }
}
streg tp* `varlist' `if' `in' , nocons d(e) `options'
end

/*****/
/* Settings and file of birth records */
/*****/

```

```

clear
#delimit;
use "m_mics_childmortality.dta";

/*****
/* Variables */
*****/

    generate RecNo = _n;
/* Person died - this is the event studied */
generate ChildDied = 0;
replace ChildDied = 1 if M_DEATH != .;
/* Observation time */
generate DUR = ( M_INTERV - M_BIRTH ) / 12.0;
replace DUR = ( M_DEATH - M_BIRTH ) / 12.0 if ChildDied == 1;
/* Censor at age 5 and drop cases with death after age 5 */
replace ChildDied = 0 if DUR > 5;
replace DUR=5 if DUR>5;
/* Mothers age group at birth */
gen AgeGrMo = 2;
replace AgeGrMo = 1 if ( M_AGEMO / 12.0 ) < 17.0;
replace AgeGrMo = 0 if ( M_AGEMO / 12.0 ) < 15.0;
char AgeGrMo[omit]2;
/* Mother graduated from primary school */
gen MothNoPrim = 0; replace MothNoPrim = 1 if M_EDUCMO < 2;

    /*****
/* Episode splitting */
*****/

stset DUR, failure(ChildDied) id(RecNo);
/* splitting file in period episodes */
stsplitt After1980, after at ( 0 10 20 30 ) ( time = 1980 - (1900 + M_BIRTH /12.0) );
replace After1980=4 if After1980==30;
replace After1980=3 if After1980==20;
replace After1980=2 if After1980==10;
replace After1980=1 if After1980==0;
replace After1980=0 if After1980==-1;
label define After1980
    0 "Before 1980"
    1 "1980-1989"
    2 "1990-1999"
    3 "2000-2009"
    4 "2010+";
label values After1980 After1980 ;
char After1980[omit]4;

    /*****
/* Estimation */
*****/
xi: stpiece i.After1980 i.AgeGrMo i.MothNoPrim i.M_MALE, tp( 1, 2, 3, 4 );

```

3.5. Migration refinements

3.5.1. Discussion

Analysis shows higher rates of migration for more highly educated people, with different magnitudes of effects by province. The option to simulate migration by education is added in this modeling phase. Micro-simulation can be a powerful tool for the detailed simulation of migration decisions on the individual and family levels. Such models can include contextual information on regions (e.g., public infrastructure, such as the availability of schools) as well as regional events like droughts and climate change. From that perspective, the presented models can be starting points for more detailed, policy-relevant modeling.

3.5.2. Parameters

Two parameters were added to the base module:

- A selection parameter allows a choice between the base module and the refined module modeling migration by education
- A table of migration probabilities by education, sex, age group, and province

Tables used from the base module are:

- The on/off switch, to model migration
- A table of migration probabilities by sex, age group, and province (applied if the base module is selected by the user)
- The distribution of destination provinces by sex, age group, and province of origin

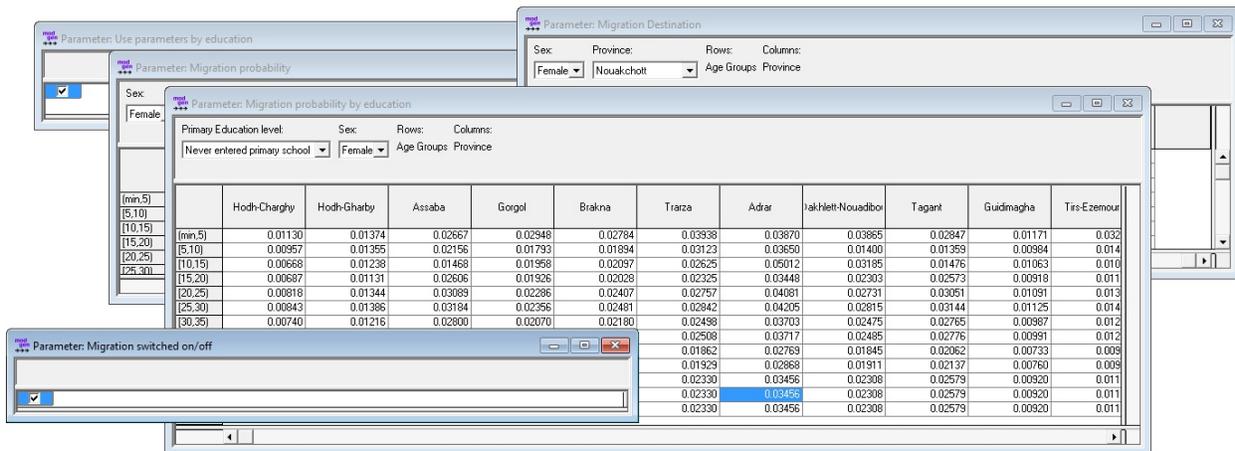


Figure 3-12: Internal Migration Parameters Including Internal Migration by Education

3.5.3. Analysis

We applied logistic regression models estimated separately by education group and sex. The impact of education is very different by province; for example, in the capital, Nouakchott, migration rates are very low and independent of education. For other provinces, migration rates are up to five times higher for primary school graduates compared to people who never entered primary education. When mechanically applied in the simulation, this effect will both accelerate the growth of Nouakchott and influence the

educational composition of its population.

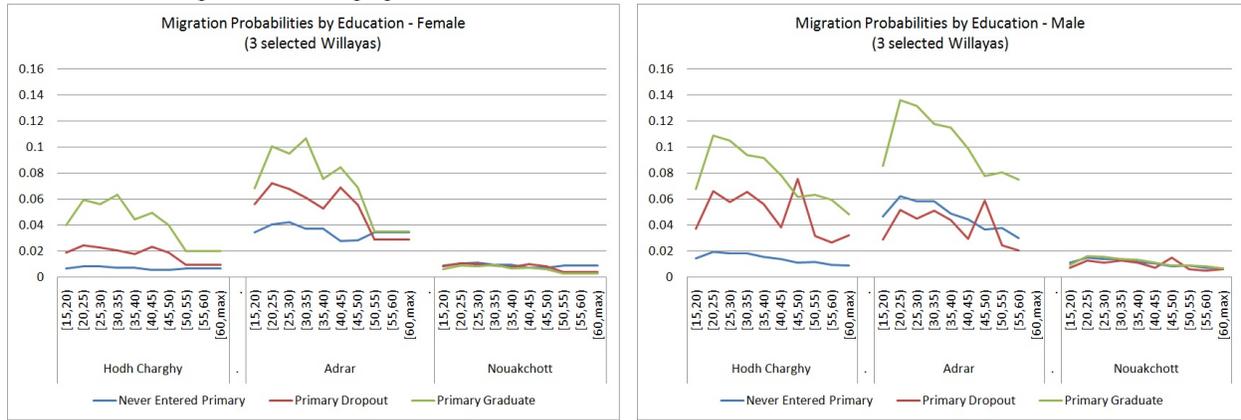


Figure 3-13: Internal Migration

3.5.4. Reproducing the Analysis

- Stata code: 17_MigrationRefined.do
- Excel parameter file: Refined_Migration.xlsx

Chapter 4. Using the model

This chapter demonstrates the use of the model by means of three examples. The first focuses on the effect of education on demographic behaviors as well as future size and educational composition of the population. We compare different educational scenarios, with the principal scenario based on past trends, and alternative scenarios that introduce universal primary education immediately or phase it in over a decade. The second example focuses on child mortality and demonstrates how micro-simulation can disassemble effects—the reduction of child mortality due to educational expansion—into their driving factors: higher education of mothers, avoidance of early teenage pregnancies due to later union formation, and a general reduction in the number of births. The third example studies the influence of education on internal migration and the future educational composition and population size of the capital, Nouakchott, compared to the rest of the country.

4.1. Analysis Example: Demographic Effects of Education

4.1.1. Scenarios

The starting point of this analysis is a **base population projection scenario**, similar to national projections provided by the Mauritanian National Statistical Office. The differences involve:

- Internal migration, which is not modeled by ONS. As will be shown later, internal migration is very sensitive to assumptions about the impact of education on migration. The base scenario used here does not include education in the modeling of migration and keeps internal migration patterns by age, sex, province of residence, and destination constant over time. Like ONS, we do not model international migration in the base scenario. When adding international migration to the model, assuming constant emigration rates by age and sex and a one percent increase in immigration over time (excluding recent refugee immigration), the population would increase faster, as shown in the comparison of three scenarios that follows.
- Fertility is modeled by education, union status, province, and parity and applies recent time trends by birth order. Interestingly, we achieved almost identical numbers of births as in the macro model and thus do not use the alignment options provided by the model. Unlike the macro model, we included fertility for ages under 15. Unlike the macro scenario, fertility changes when education assumptions are changed. Education affects fertility both directly (more highly educated women have lower fertility) and indirectly, via changes in union formation age (more highly educated women have a higher average age at first marriage).
- While we use the same age-specific mortality rates as in the macro projection, we “activated” the child mortality module, which applies relative risks to child mortality (at ages 0-4) by mother’s education and age. While the child mortality model is calibrated to the overall age-specific mortality rates at the beginning of the simulation and applies the same overall improvements in mortality

rates as in the macro model, the changing education composition of mothers leads to a faster decline in mortality rates than in the macro model. Child mortality issues are discussed in more detail in the next chapter.

PROJECTED TOTAL POPULATION - MIGRATION SCENARIOS

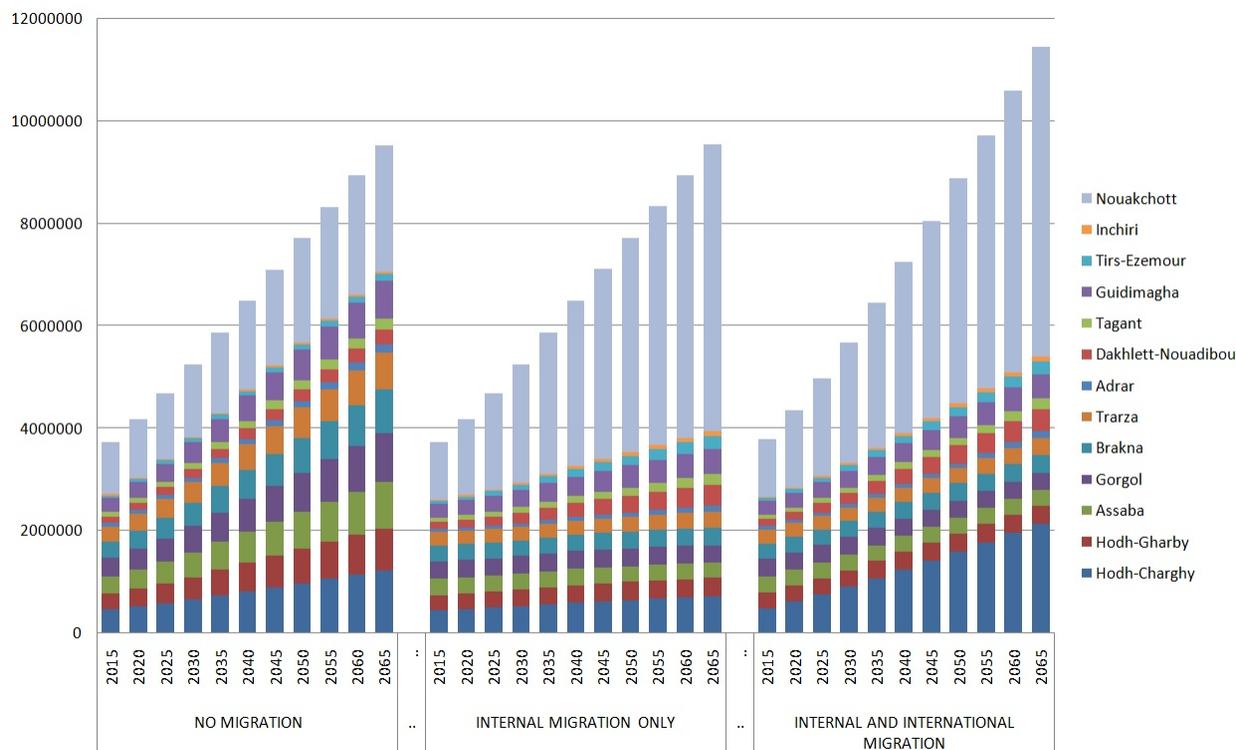


Figure 4-1: Population by Province and Migration Scenario

In Figure 4-2, we use the “internal migration only” scenario as the base scenario.

For education, the base scenario uses the model discussed in detail in *Chapter 3, section 3.3*. The model is based on past trends of entry and graduation probabilities in primary education. They are modeled by sex, year of birth, and province of birth, assuming constant relative differences between provinces (expressed in odds ratios) as observed in the past. Provincial differences are very pronounced: in Nouakchott, 60 percent of 15-year-olds have graduated primary school, while in the second largest province, Hodh Charghy, this number is only 20 percent.

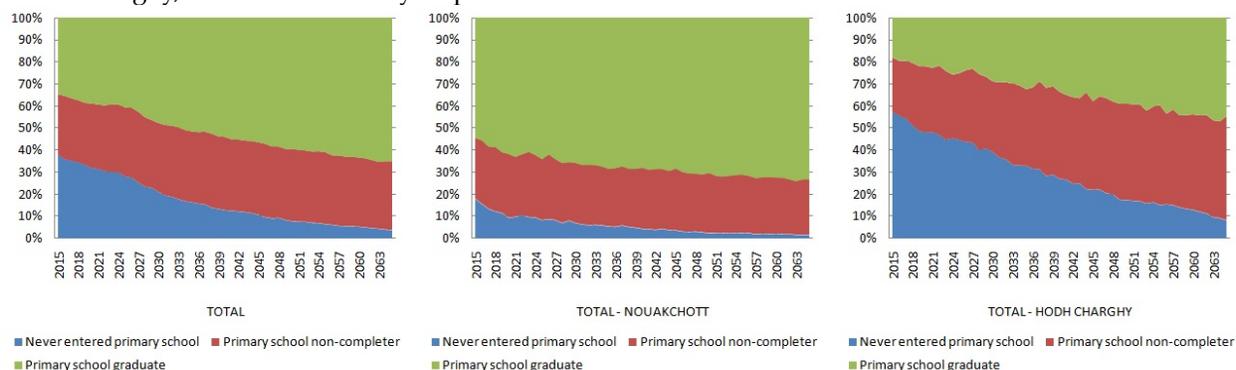


Figure 4-2: Primary Education of 15-year-old. National and Provincial Examples. (Simulation Results)

In an alternative scenario of universal primary education, we assume universal primary education is

effective for all children born during and after 2010. For some analysis, we also add a **scenario of phased-in universal primary education**, where universal primary education is reached linearly over a period of 10 years, starting with the 2010 birth cohort.

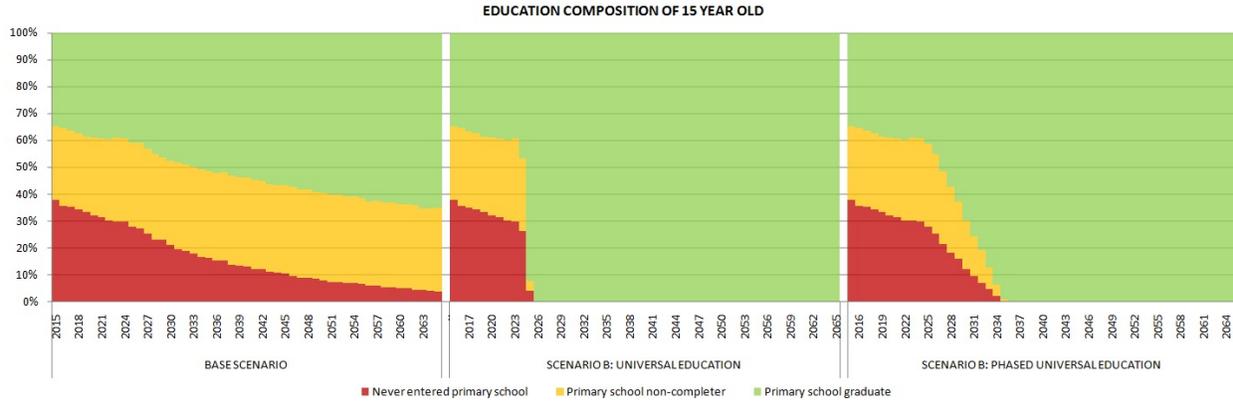
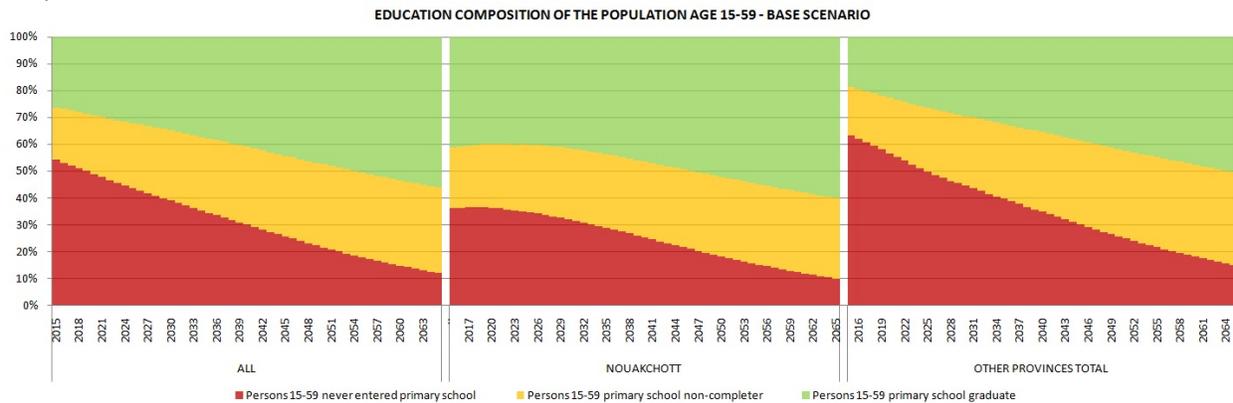


Figure 4-3: Primary Education of 15-year-old. Alternative Scenarios. (Simulation Results)

4.1.2. Results

As the current young and future cohorts affected by educational improvements replace the overall population over time, the educational composition of the total population also changes. The speed of change depends on the size and speed of the educational improvements, and is fastest in the “universal primary education scenario.” The following three graphs show how the educational expansion changes the education composition of the population age 15 through 59. Given the current very high educational differences between provinces, both the starting point and speed of improvements are very different on a regional level. In the figures, we contrast the capital, Nouakchott, with the rest of Mauritania. Note that the provincial educational distributions also result from internal migration. The impact of education on migration and the resulting educational compositions of populations by province will be discussed in detail in *Chapter 4, section 4.3.*



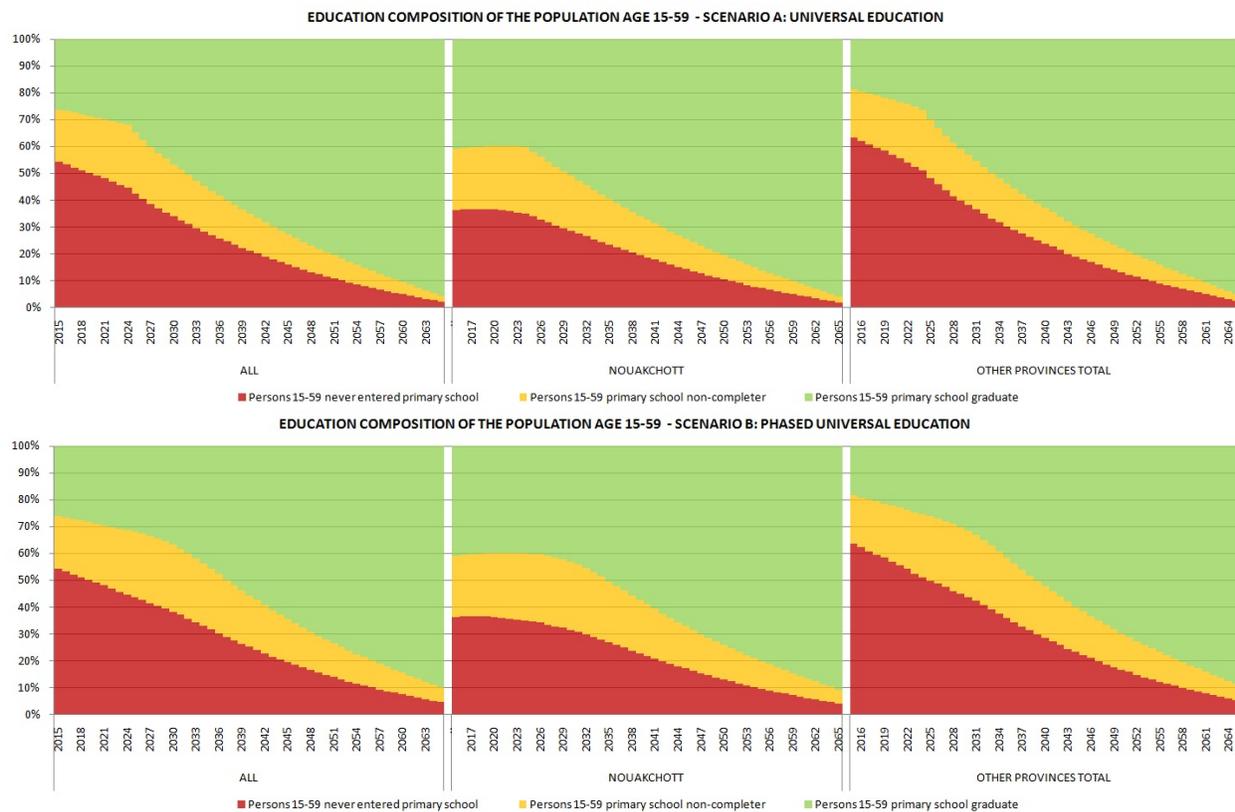


Figure 4-4: Projected Educational Composition of the Population Age 15-49 in Three Scenarios

Improvements in education impact not only the educational composition of the population, but also the future size as more highly educated women, on average, give birth later and have fewer children.

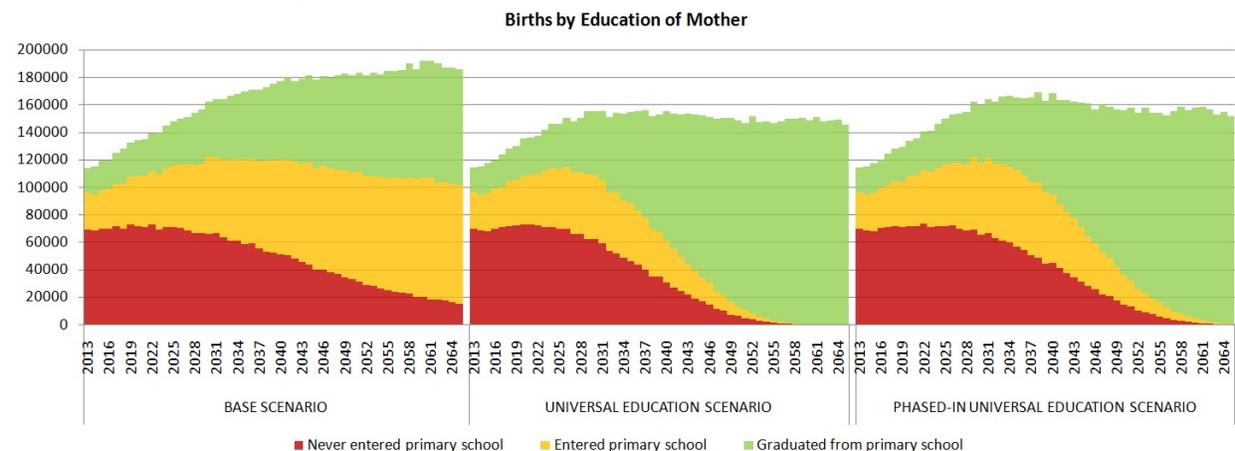


Figure: Projected births in three scenarios

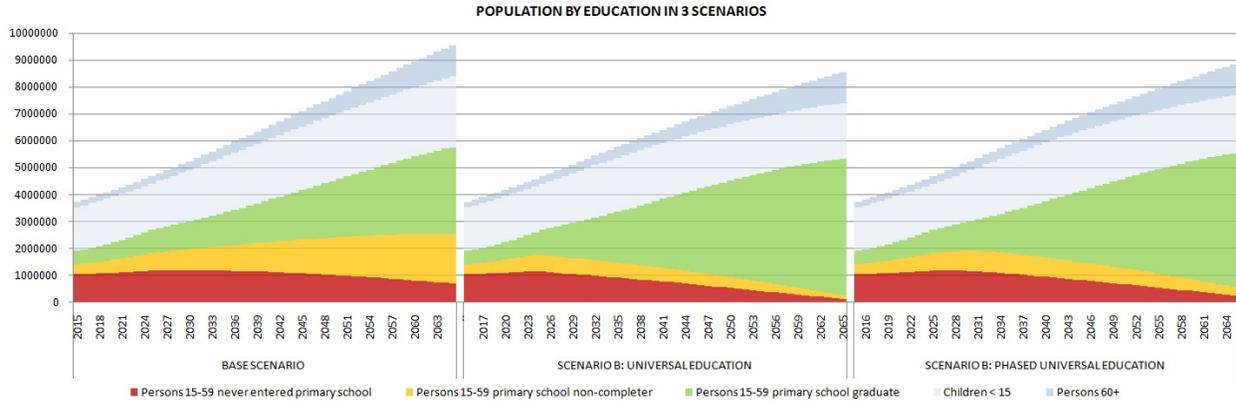


Figure 4-5: Projected Total Population in Three Scenarios

Fewer births also result in fewer deaths. Education’s impacts on mortality, and specifically the impact of mother’s education on child mortality, are detailed in Chapter 5. In this chapter, we discuss the various demographic impacts of education, comparing base to universal primary education scenarios.

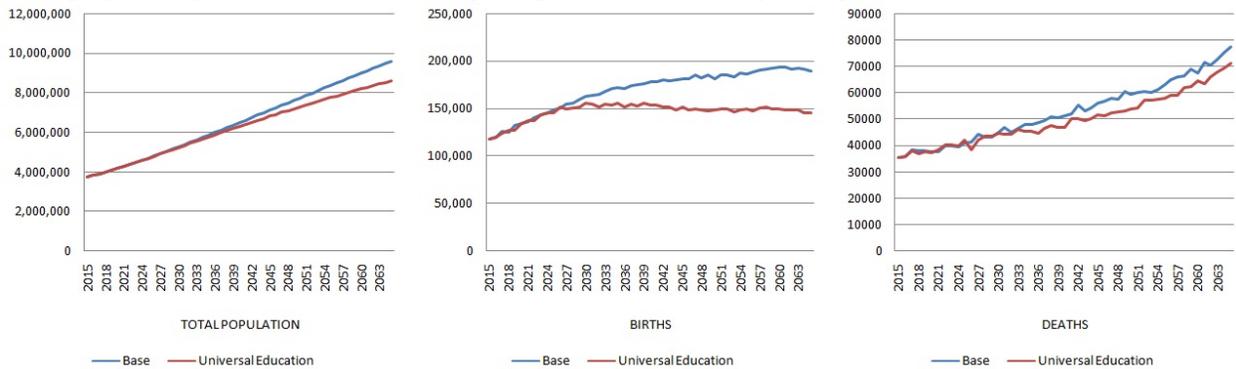


Figure 4-6: Total Population, Births, and Deaths in Two Education Scenarios

The most striking result of the universal education scenario is the flattening of the total number of births, plateauing at a level of 150,000 births per year. As shown in Figure 4-7, this has a pronounced impact on the future age composition of the population (the age pyramid).

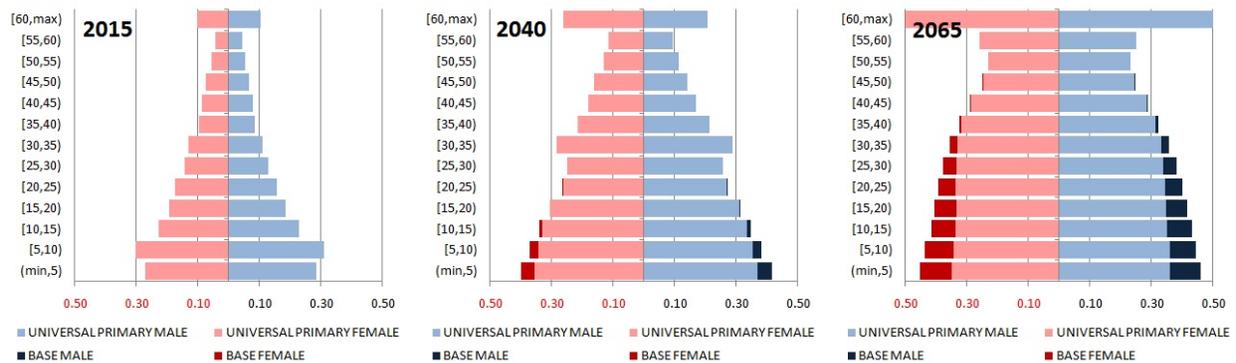


Figure 4-7: Age Pyramids 2015, 2040, 2065 in Two Education Scenarios

An immediate effect of universal education is the reduction in the number of early teenage pregnancies when the first cohorts of females with universal primary education reach reproductive age. Births by mothers under age 17 drop to less than half, compared to the base scenario. Given the effects of low levels of education and youth on child mortality, we also project a drop in infant and child deaths. Another effect of educational improvement is an increase in the average age for first marriages.

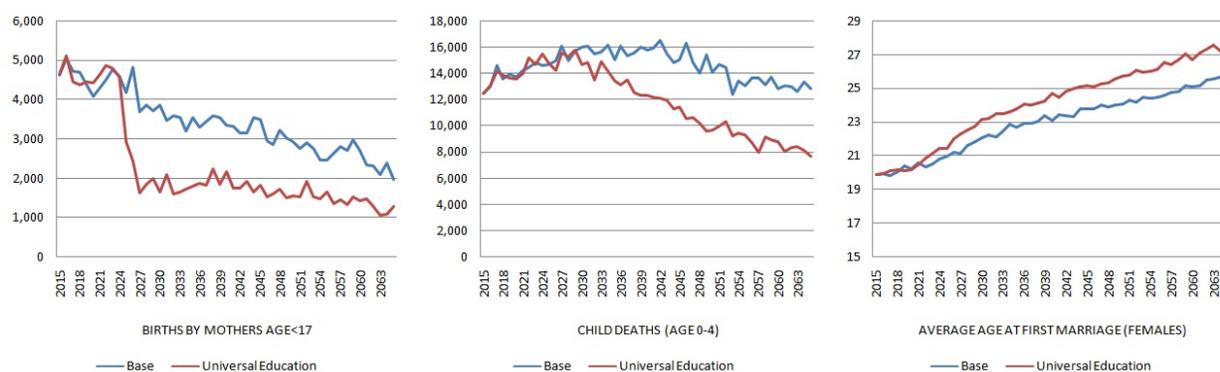


Figure 4-8: Number of Early Teenage Births, Child Deaths, and Average Age at First Marriage

4.2. Analysis Example: De-Composing Changes in Child Mortality

4.2.1. Scenarios

In this analysis, we contrast two main scenarios:

- The **base scenario** models educational improvements based on past trends, and is the same education scenario described in *Chapter 3, section 3.3.* and *Chapter 4, section 4.1.*. To better understand mortality effects stemming from educational change rather than overall improvements in mortality rates, we use a status quo model in mortality with no time trends. While we apply relative risks by age and education of mothers, there is no trend in the baseline rate, which is calibrated to produce the same current overall mortality as the macro model. Over time, the composition of mothers by age and education changes and overall mortality will also improve in the base scenario. This improvement stems entirely from composition effects, i.e., the improved education of mothers.
- The **alternative scenario of universal primary education** assumes that all children born in and after 2010 enter and complete primary education. This is the only difference compared to the base scenario.

To calculate the individual contributions of changes in single processes to the overall changes in mortality outcome, we ran additional scenarios in which the relative risks by mothers' education and/or mothers' age are removed.

At the beginning of the simulation, the overall mortality is calibrated to the assumptions of the macro model used by the Mauritanian ONS. The calibration is performed automatically by the micro-simulation model if the related option is activated:

- ~ 0.075 at age 0 (averaging male and female mortality)
- ~ 0.012 at ages 1-4 (averaging male and female mortality)

The key settings are visible in the following screen-shot of parameters:

- We chose the initial calibration of mortality with application of specific child mortality trends instead of the overall trends calculated from life expectancy parameters.
- We set the trend parameters to 1 over all calendar years, thus not assuming any overall calendar time trend.
- The relative risks are left unchanged from the model described in *Chapter 3, section 3.3.*

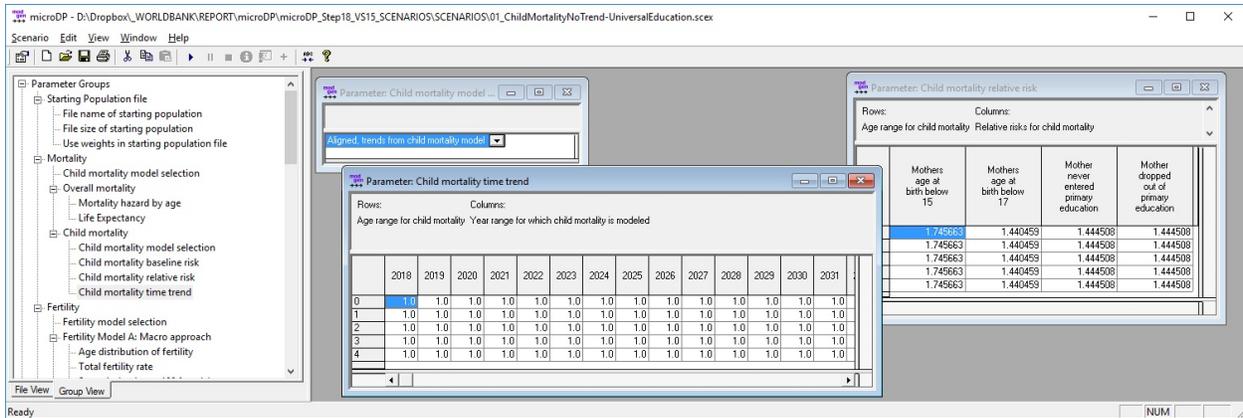


Figure 4-9: Key Parameters for Child Mortality Assumptions

4.2.2. Results

As education improves in both scenarios, overall mortality drops over time. This improvement is, of course, higher in the alternative scenario, which immediately makes primary education universal. Most of the improvement in mortality rates can be attributed to the higher education of mothers, rather than the decrease in early teenage pregnancies. This can be demonstrated by running an additional scenario with no relative risk by mother’s age. The result is intuitive, as the child mortality improvement in more highly educated mothers applies to the whole lifespan of women respectively. and all the children they have, and not to children born in early teenage years, which become very rare due to the improvement in education.

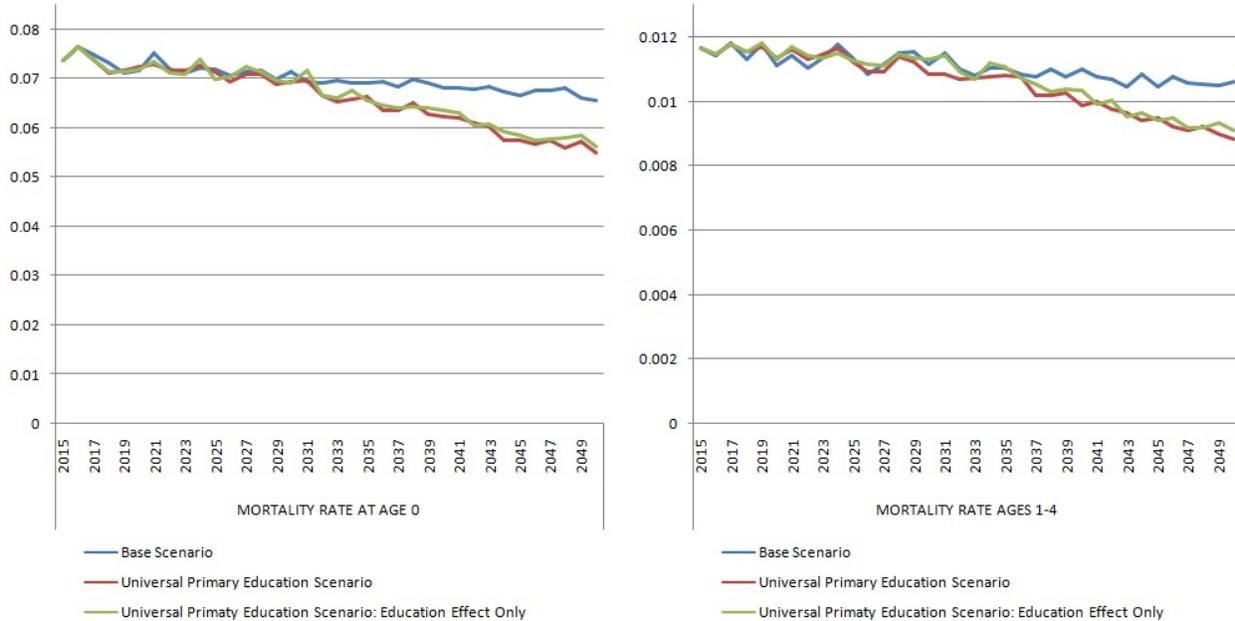


Figure 4-10: Reduction of Infant and Child Mortality Rates Due to Introduction of Universal Primary Education

When looking at numbers of child deaths instead of rates, an additional effect stems from the lower number of births as more highly educated women, on average, give birth later in life and have fewer children. Figure 4-11 compares the number of deaths between the two education scenarios for three calendar year periods. As expected, the effect increases over time as more highly educated cohorts of women replace the population in the fertile age range. In an additional step—by creating and comparing scenarios in which we remove single relative risks—we can separate the decrease in child deaths into their main causes: the higher education of mothers, which is the greatest factor; the avoidance of early

teenage pregnancies, which has the least effect; and the overall reduction in the number of births, which is responsible for about one-third of the total reduction in child deaths.

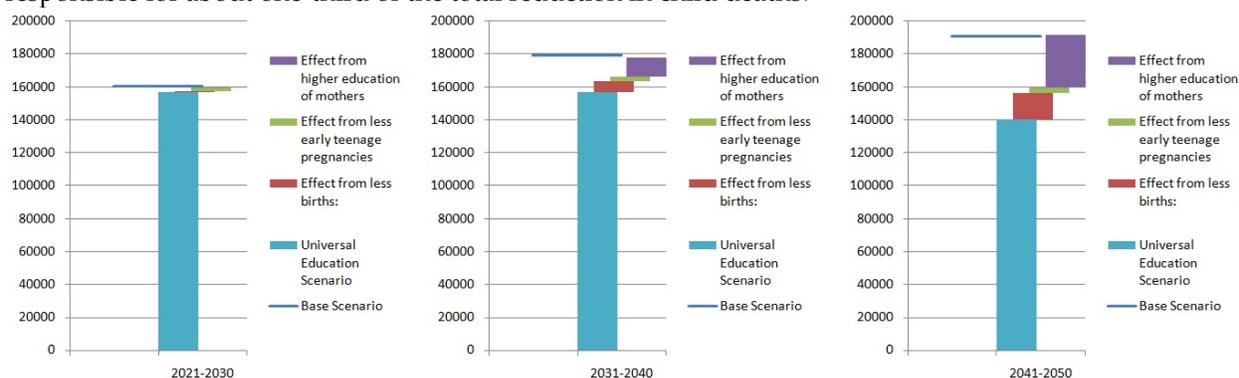


Figure 4-11: Decomposition of the Reduction of the Number of Child Deaths at Ages 0-4 Due to Universal Primary Education

4.3. Analysis Example: Internal Migration by Education

In this example, we focus on the sub-national level of simulation results. From a provincial perspective, the future size and age composition of the population depends heavily on internal migration. Given the inter-provincial differences in educational attainments, internal migration also impacts the education composition of the population within provinces. For example, in the Mauritanian case, we observe high numbers of migration to the capital, Nouakchott, which has comparable high school attendance rates. The way in which migration from provinces—which, on average, have lower education—affects the educational composition Nouakchott’s population will depend on the extent to which educational attainment affects migration probabilities. Is it those with greater or less education who migrate? In this analysis, we compare the size and educational compositions of the population of Nouakchott to that of the rest of the country.

4.3.1. Scenarios

For this analysis, we compare five scenarios. The first, A0, is a scenario without internal migration. Two scenarios, which involve internal migration modeled by age and sex under two alternative assumptions on educational expansion, are based on scenarios introduced in the previous chapters. In the two remaining scenarios internal migration is modeled by age, sex, and education.

- To be able to assess the overall impact of internal migration, we created a **Scenario A0** without internal migration. For education, we continued educational trends as observed in the provinces.
- For **Scenario A**, we combine the base model option of migration with the base assumptions on education, which continues educational trends as observed in the provinces.
- In **Scenario B**, we combine the base model option of migration with a scenario of universal primary education for all birth cohorts since 2010.
- For **Scenario C**, we select the refined migration model, which also accounts for educational differences in migration rates. The education scenario here is the base trend scenario.
- **Scenario D** combines the refined migration model with the assumption of universal primary education.

As discussed in *Chapter 3, section 3.6*, analysis shows higher rates of migration for more highly educated people, with different magnitudes of effects by province. It must be noted, however, that the projections

assume that these current observations also hold for the future. While such assumptions are useful for studying the sensitivity of migration to educational assumptions, they ignore causal mechanisms of migration, such as availability of educational institutions or differences in job opportunities by education. These factors, like many others affecting migration, may change in the future. Micro-simulation is not limited to such simple assumptions, but can provide a powerful tool for the detailed simulation of migration decisions both on the individual and family level. Such models can include contextual information on regions (e.g., public infrastructure, such as the availability of schools) or regional events, such as droughts and climate change. From that perspective, the models can be starting points for more detailed and policy-relevant modeling.

4.3.2. Results

Zero Migration Scenario A0

In the absence of migration, population grows over-proportionally fast outside Nouakchott. Nouakchott distinguishes itself with a substantially higher proportion of primary school graduates, which is one of the factors leading to lower birth rates. Starting with the lower ages, this alters the shape of the age distribution from a typical age pyramid to a “column.” In 2065, cohort sizes of 0 to 25 years old are very similar. In contrast, the rest of the country keeps its current education shape, driven by high fertility.

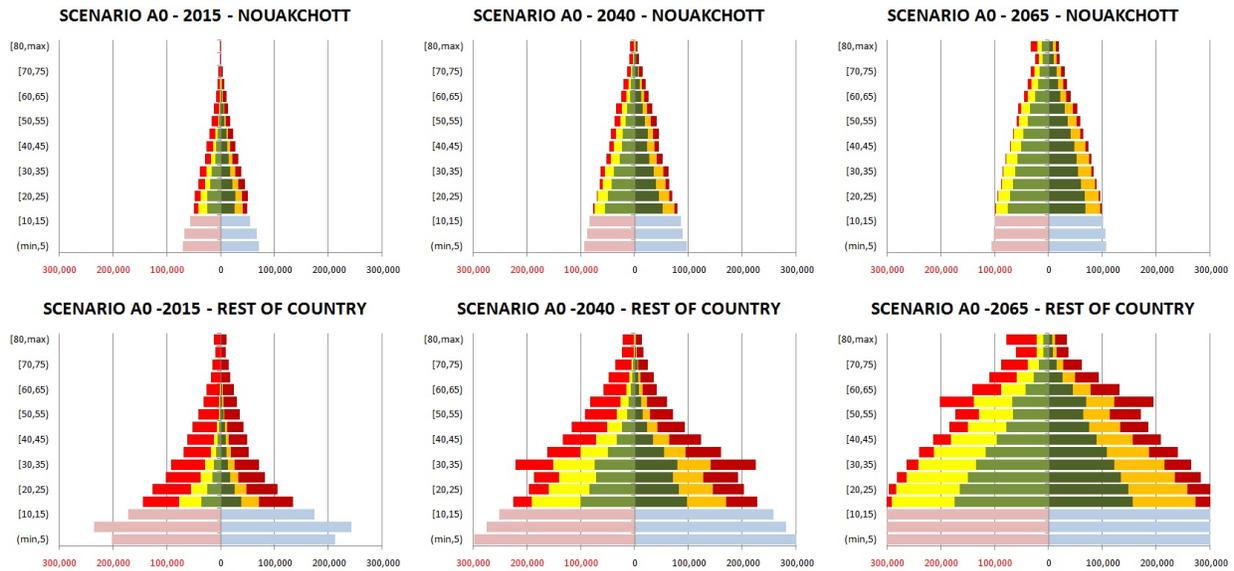


Figure 4-12: Projected Population in Scenario A0. Legend: left female, right male; green: primary graduates, orange/yellow: primary dropouts, red: never entered primary.

The Base Scenario A: Continuing Educational Trends, Base Migration (Without Education)

When internal migration is added to the projection, results are very different: Nouakchott now grows over-proportionally fast and part of its comparably favorable educational composition is lost due to the proportionally lower education of in-migrants. At around 2040, Nouakchott reaches the same population size as the rest of the country with very similar age distribution.

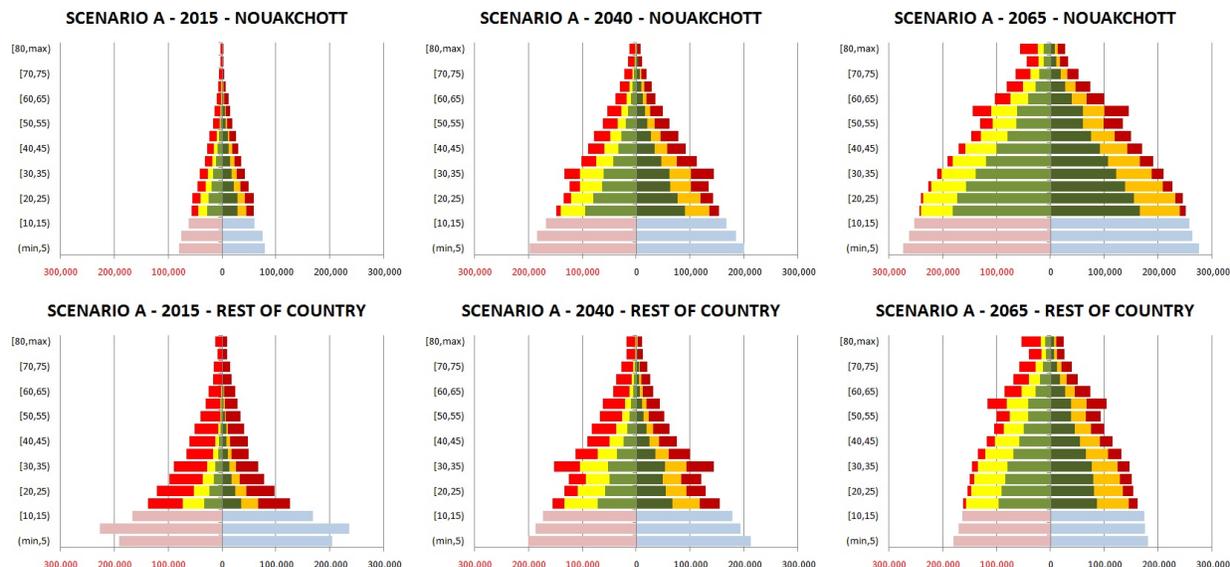


Figure 4-13: Projected Population in Scenario A. Legend: left female, right male; green: primary graduates, orange/yellow: primary dropouts, red: never entered primary.

Scenario B: Universal Primary Education, Base Migration (Without Education)

In a scenario of universal primary education attained by all birth cohorts 2010 and later, demographic effects are relatively small until 2040. As women of reproductive age are increasingly dominated by primary graduates, however, birth rates drop, and in Nouakchott and the rest of the country, the age pyramids start transforming into “columns” with very similar cohort sizes.

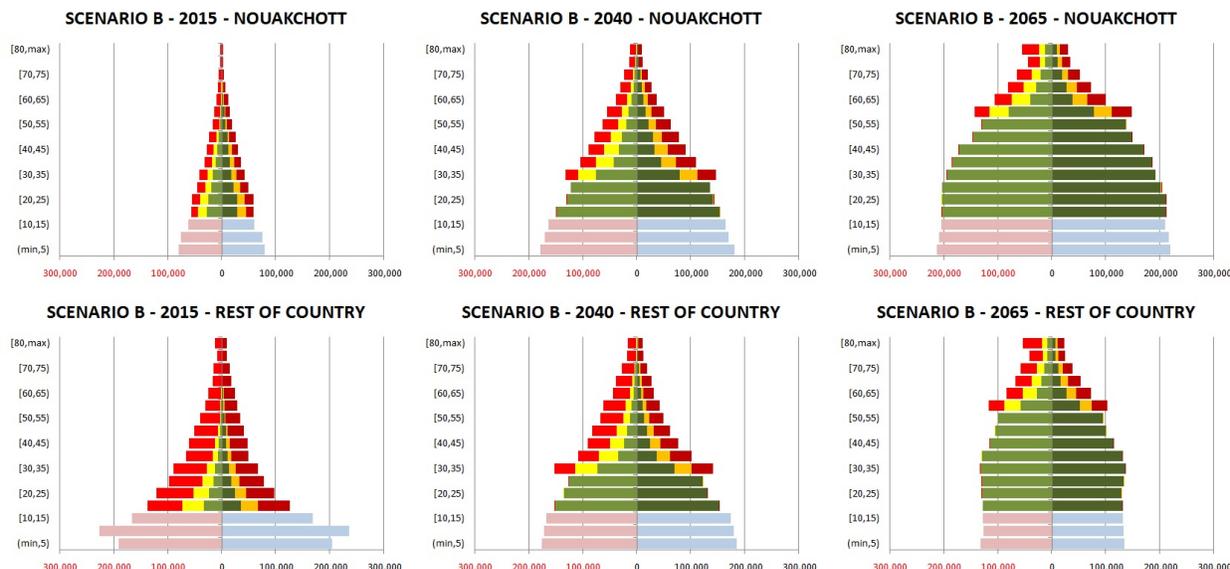


Figure 4-14: Projected Population in Scenario B. Legend: left female, right male; green: primary graduates, orange/yellow: primary dropouts, red: never entered primary.

Scenario C: Continuing Educational Trends, Refined Migration (With Education)

In this scenario, we use the refined migration module, which accounts for education in migration rates. As we observe higher migration rates for more highly educated people, Nouakchott grows even faster than in the base scenario. In contrast to the base scenario, the education composition of the capital’s popu-

lation comes closer to the scenario without migration and the more favorable education composition is kept, as the in-migrants now do not differ significantly in education composition from the native population.

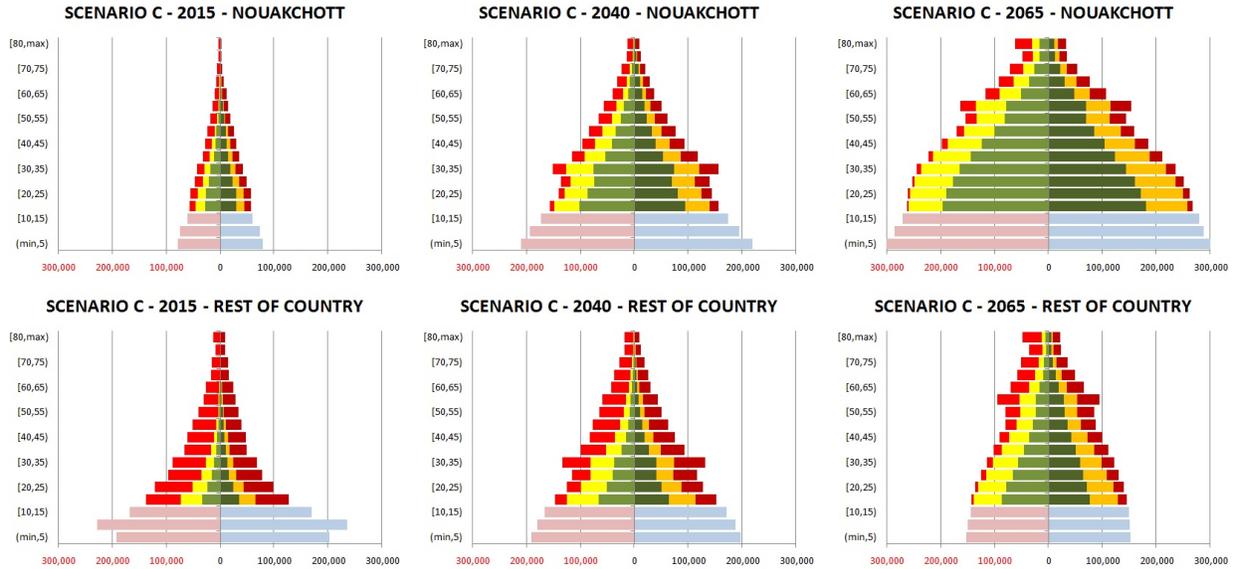


Figure : Projected Population in Scenario C. Legend: left female, right male; green: primary graduates, orange/yellow: primary dropouts, red: never entered primary.

Scenario D: Universal Primary Education, Refined Migration (With Education)

When the refined migration module is combined with a scenario of universal primary education, it erases the educational differences between Nouakchott and the rest of the country, as eventually all graduate from primary school. While migration rates and the migration-driven growth in the capital increase, birth rates drop, which leaves the total population of the capital almost unaffected, while increasing its average age. In contrast, population in the rest of the country shrinks, and those born after 2010 are more likely to migrate to the capital, due to their universal primary education. Both high migration and universal primary education make the population over-proportionally subject to population aging—very much in contrast to the initial scenario without migration.

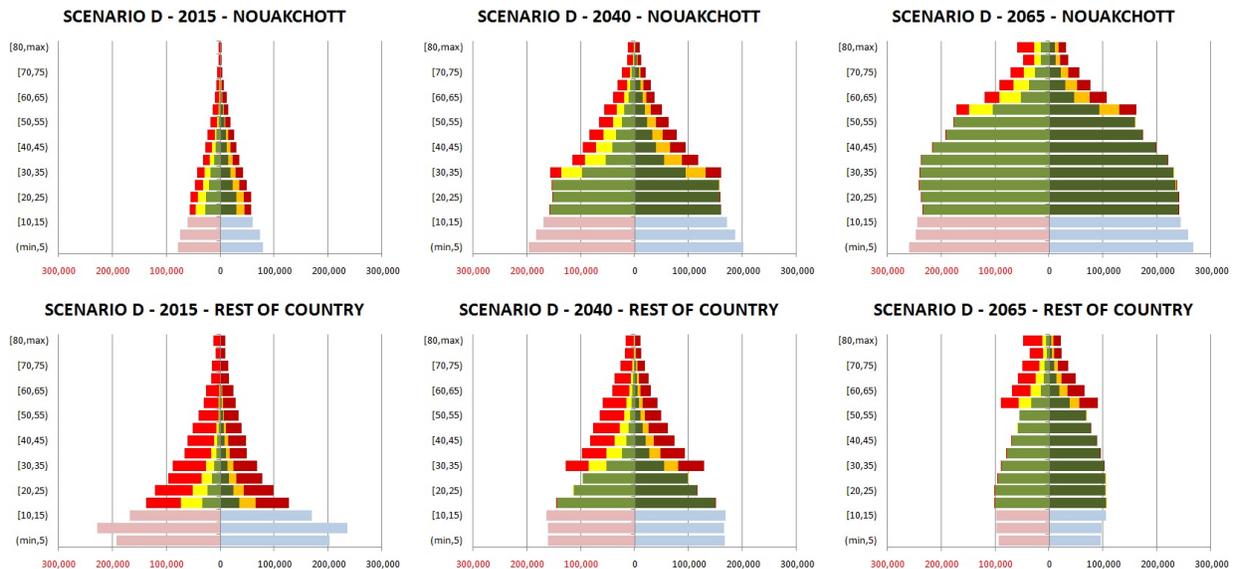


Figure 4-16: Projected Population in Scenario D. Legend: left female, right male; green: primary graduates,

orange/yellow: primary dropouts, red: never entered primary.

Chapter 5. Micro-simulation Technology: Modgen/OpenM++

Dynamic micro-simulation models can be implemented in three ways: from scratch, using multiple purpose programming languages like C++; using (and extending) statistical packages (e.g., R) with or without combining other tools; and using specialized micro-simulation packages. In this report, we advocate and use the latter approach, specifically, Modgen, a generic micro-simulation technology and language developed at Statistics Canada. Compared with other approaches, using Modgen allows considerable efficiency gains, as it avoids re-inventing the wheel (still a common practice in micro-simulation), allows communication between a large and growing global user group, and, as an industrial-strength product, avoids many programming problems and bugs because many components are ready made and well tested. As a compiled language, Modgen is also very fast, so large-scale models can be implemented with millions of actors that can be run on standard PCs or computer networks.

Modgen (Model generator) is a generic micro-simulation programming language that supports the creation, maintenance, and documentation of dynamic micro-simulation models. Virtually all types of dynamic socioeconomic and sociodemographic micro-simulation models can be accommodated, from small and specialized to large and multi-purpose, in continuous or discrete time, with interacting or non-interacting populations. Furthermore, a model developer does not need advanced programming skills to use Modgen, because Modgen hides underlying mechanisms, e.g., event queuing, and creates a stand-alone model-executable program with a complete visual interface and detailed model documentation. Model developers can therefore concentrate on model-specific code: the declaration of parameters, simulated actors, and events. High-efficiency coding, typically requiring only a couple of lines, can be used to program an output table. Tabulation is done in continuous time and includes a mechanism for estimating the Monte Carlo variation for any cell of any table.

Modgen was designed to facilitate micro-simulation model programming, and its purpose is to remove obstacles to micro-simulation model creation. Obstacles that Modgen eliminates include interface programming, documentation, simulation engine programming, and multi-lingualism. Modgen also provides the interface and simulation engine, and facilitates documentation by automatically creating a hypertext model documentation out of the programming code, and the labels and comments documenting the code. Recent developments include the possibility to publish and run models on the web.

Modgen was developed by micro-simulation modelers and grew by accommodating concrete modeler demands in various application fields covering population projection, health models, and large-scale socio-economic models, e.g., pension analysis. Since it can be shared for free, Modgen is used internally at Statistics Canada and by modelers around the globe, including governments, academia, individual researchers, and international organizations like WHO and OECD (for a list of models using Modgen, consult www.statcan.gc.ca/eng/microsimulation). This wide range of users can also extensively test Modgen in a variety of application fields, contributing to its reliability and stability.

Recent developments in micro-simulation technologies include the development of the open source language OpenM++, a micro-simulation platform inspired by and compatible with Modgen. OpenM++, compared to its free but closed source predecessor, has many distinct features like portability, scalability,

and open source. It is currently developed in coordination with Modgen developers, and as an open source project, is expected to reach an even broader user community (http://ompp.sourceforge.net/wiki/index.php/Main_Page).

In this section, we introduce the Modgen technology from two perspectives: first, the user, then the developer. From the users' perspective, key features are a clear and easy graphical user interface, high execution speed, full documentation, the ease of modifying and creating scenarios by changing parameters or selecting modeling options, and rich model output in fully labeled tables and micro-data files. From the developers' perspective, of key importance is the ability to fully focus on the modeling aspects, as most model-independent components are provided and do not require programming.

5.1. The User's Perspective: User Interface

This chapter expands on the short presentation of the user interface in *Chapter 1, section 1.4*. Modgen models are Windows applications running on standard PCs and share the same graphical user interface, with a menu bar, short-cut icons for key commands, and a help system. The Modgen user interface is organized into two parts: a navigation area displaying a hierarchical list of all parameters, and, after running the model, all output tables. For both parameter and output tables, space is typically devoted to the display area. Parameter tables can be directly edited in the user interface. By changing parameters, users can create and save new scenarios. After running a model, all output tables become available. Table results can be exported into Excel individually or together, to an Excel Workbook.

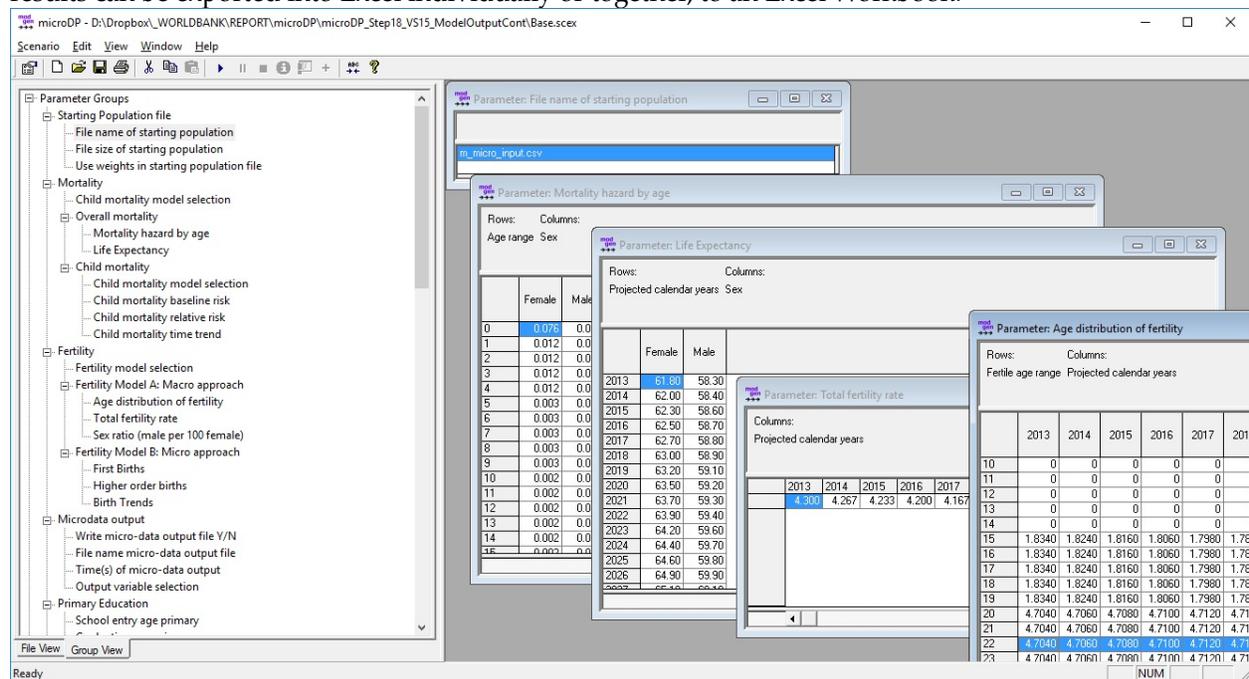


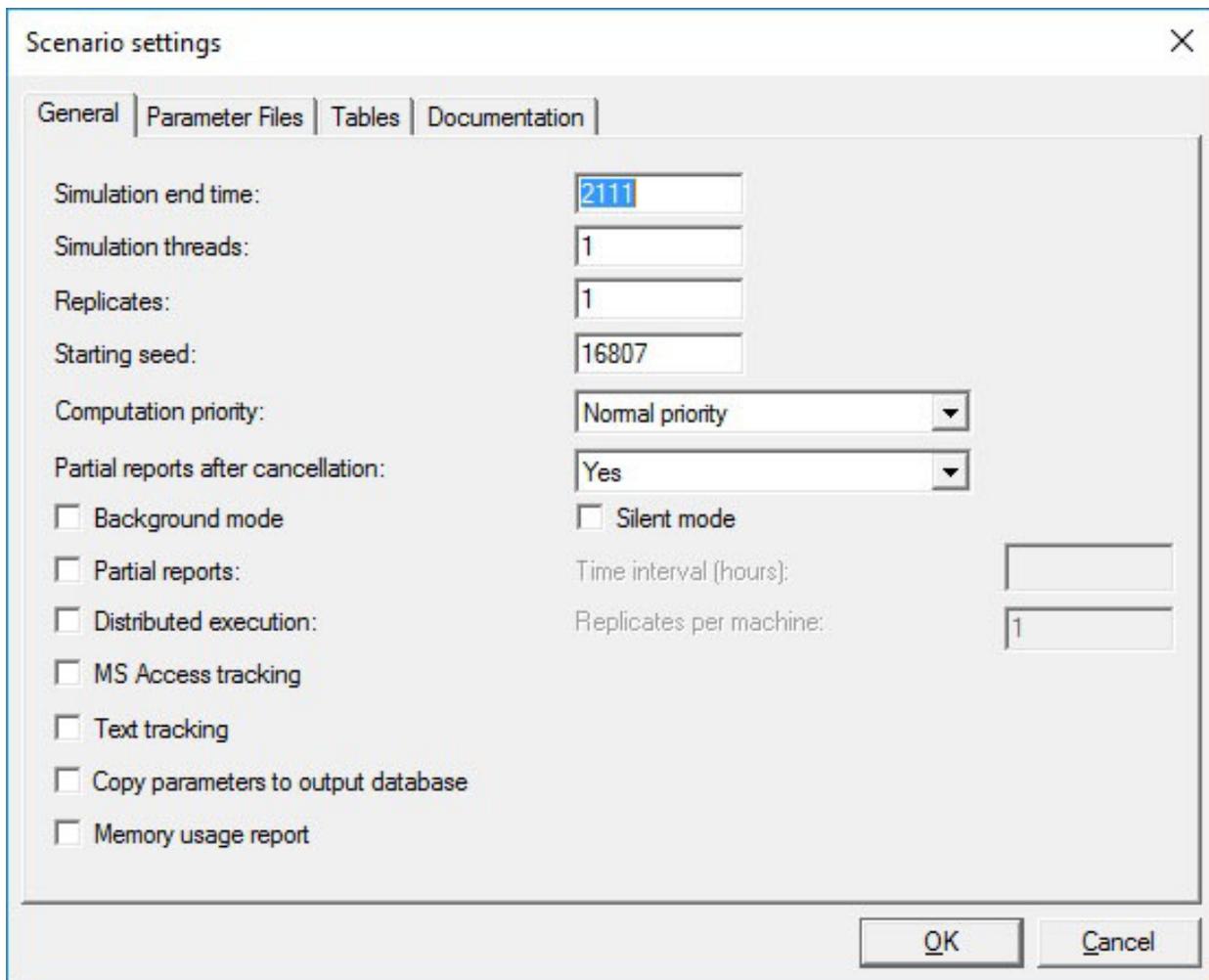
Figure 5-1: The User Interface

5.1.1. Running a model

A typical simulation analysis session will consist of the following actions:

- Opening the model: The user interface opens, both the navigation and the table pane are empty, as no scenario was opened yet.
- Selecting Scenario/Open from the menu or clicking on the open folder icon opens a scenario file. Scenario files have the extension scex.

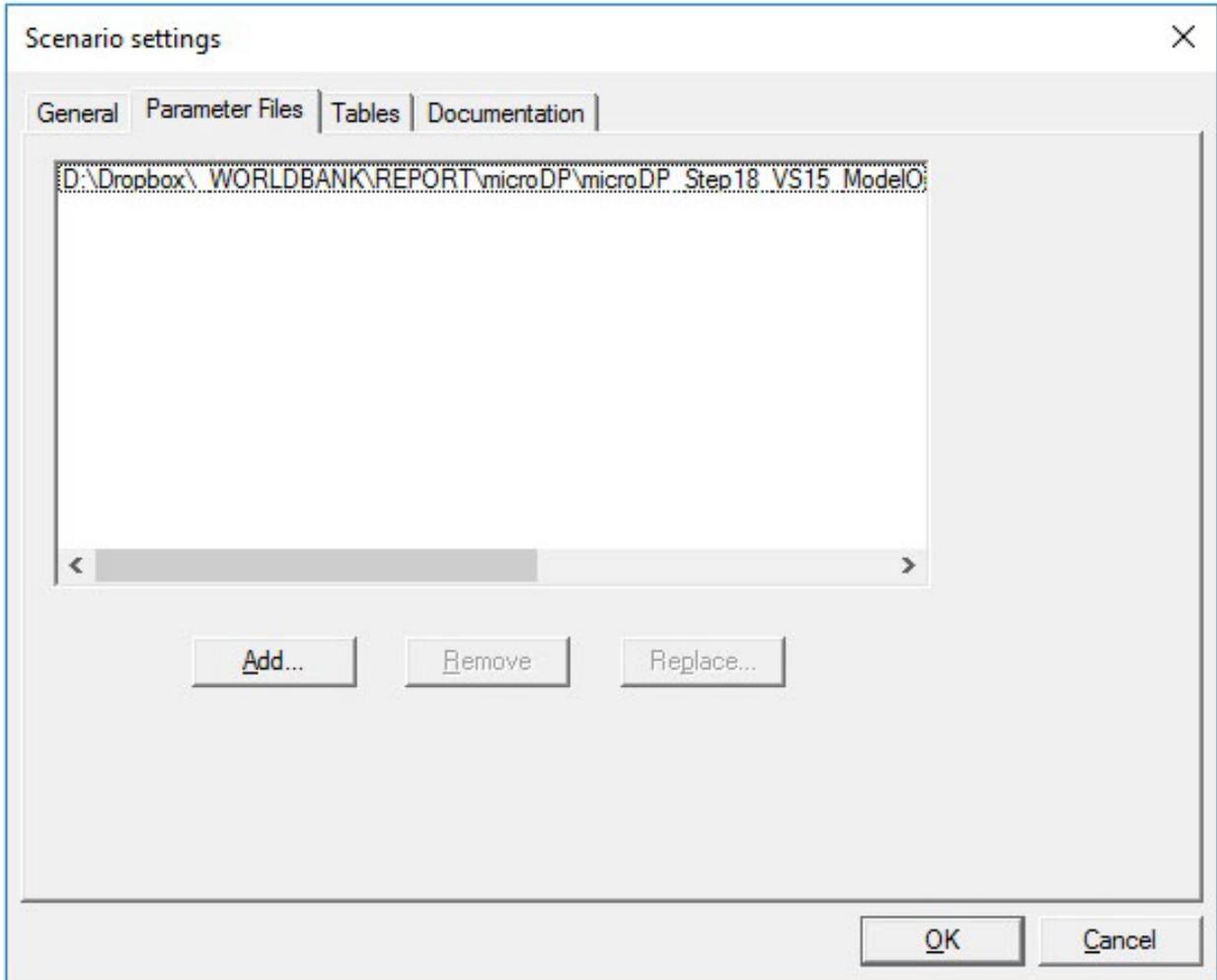
- The list of available parameters will be visible in the navigation pane. Parameters are organized in tables, which can be accessed by clicking on the table name in the navigation area. For easier navigation, the list of tables is grouped by topic.
- New scenarios can be created by changing parameters and saving the scenario under a new name. The application saves all simulation results with all parameters.
- If the scenario was run before saving, all output tables are available and can be selected like parameter tables.
- Both parameter and output tables can have any number of dimensions. Their complexity can vary from a simple, single on/off check box to multidimensional tables by age, sex, province, year, etc. The user controls which two dimensions of a multi-dimensional table are displayed and which dimensions are selected from a drop-down list. This and other table functionalities are accessible when right-clicking on a table.
- Before running a new scenario, save the existing scenario under a new name, then edit model parameters and settings.
- Users can edit tables directly, including copying and pasting from other tables, such as Excel.
- Besides parameter tables, users can also control a number of scenario settings. These are accessible by selection Scenarios/Settings or clicking the far left icon.
- After setting all parameters and settings as requested, the new scenario can be run by selecting Scenario/ Run or clicking the RUN icon.
- The updated list of model results tables is then available.



The image shows a 'Scenario settings' dialog box with a close button (X) in the top right corner. It has four tabs: 'General', 'Parameter Files', 'Tables', and 'Documentation'. The 'General' tab is selected. The settings are as follows:

- Simulation end time: 2111
- Simulation threads: 1
- Replicates: 1
- Starting seed: 16807
- Computation priority: Normal priority
- Partial reports after cancellation: Yes
- Background mode
- Silent mode
- Partial reports: Time interval (hours):
- Distributed execution: Replicates per machine: 1
- MS Access tracking
- Text tracking
- Copy parameters to output database
- Memory usage report

At the bottom right, there are 'OK' and 'Cancel' buttons.



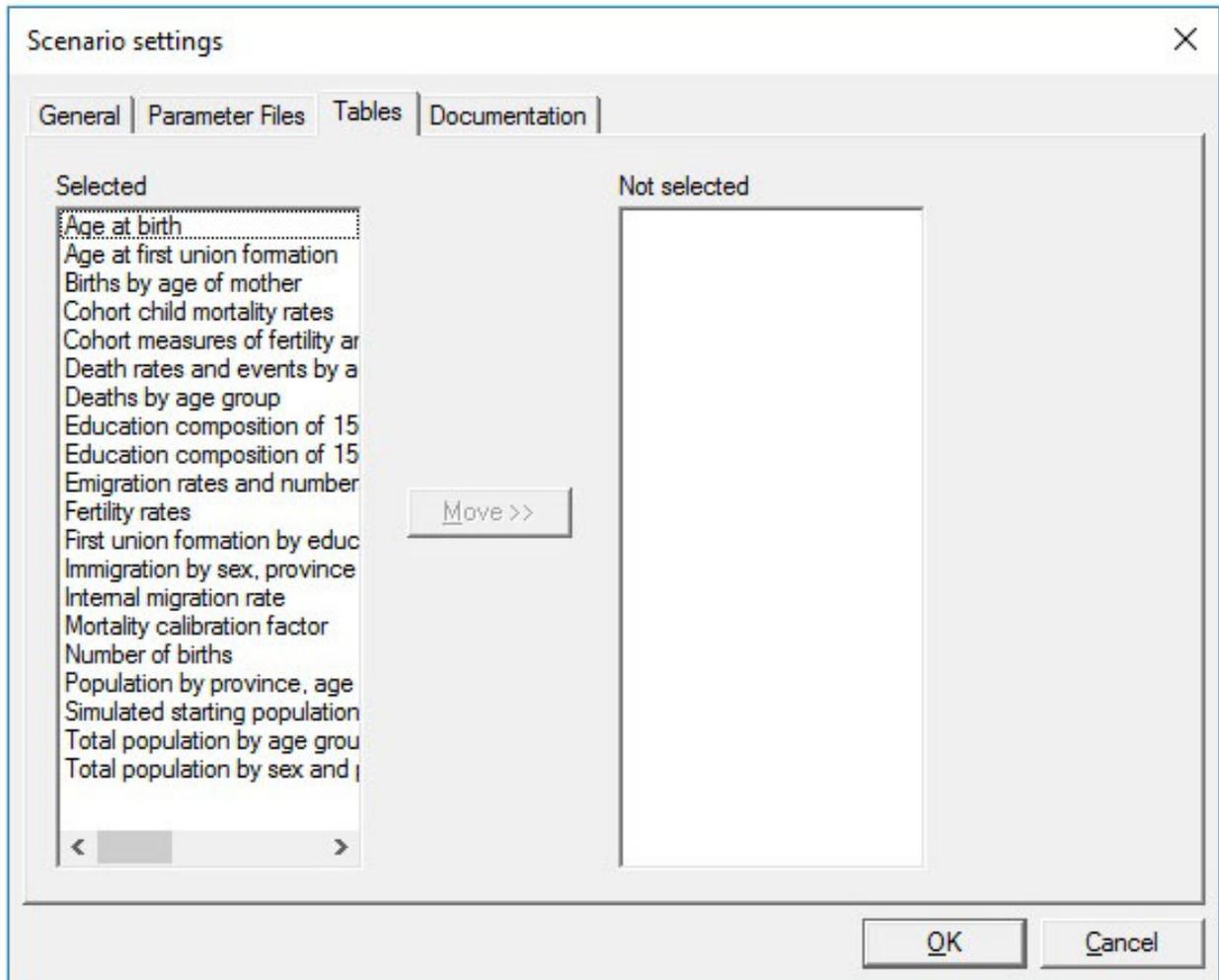


Figure 5-2: Scenario Settings

5.1.2. Scenario Settings

Important settings are:

- General tab:
 - The time horizon of the simulation.
 - The number of replicates produced: if the model is run more than once, distributional information is automatically generated from the individual model runs and results tables automatically average all results.
 - Number of threads: various replicates can be run in parallel, depending on the number of available processors.
 - Starting seed (of random numbers): when rerunning a model with the same seed, it will produce identical results.
 - Copy parameters to the output database: useful when exporting simulation results to Excel workbooks, which then can include all parameters of the model, too.

- Parameter Files tab: can see, change, or add parameter files belonging to the scenario. When saving an existing scenario under a new name, all parameter files of the model will automatically be copied with a file name consisting of the scenario name and data file name. Usually no user operations are required.
- Tables Tab: displays the list of available output tables. Tables can be de-selected if not needed.
- Documentation tab: users can place and store scenario notes here.

5.1.3. Model Results

Model results come in three forms:

- A list of tables displayed directly within the application. Besides the table values, tables can contain additional views (e.g., of distributional information of table cells) and functionalities. With multi-dimensional tables, users control the order in which to display table dimensions and details like the number of decimal places. Tables can also contain documentation. Tables are labeled, adding to the readability of the output.
- Micro-data files to be further analyzed using statistical software (depending on the model). The population projection model for examples can produce periodic files (the user sets the first time point of file output and the time interval for repeated output) of a list of variables, which the user selects. Output is written into CSV text files, which have a header row with variable names.
- A sample of individual life-course data, which can be displayed graphically using the BioBrowser application. This feature is not used in the population projection model, but is very useful, especially for debugging, as it can filter “strange” cases and study simulated lives in full detail.

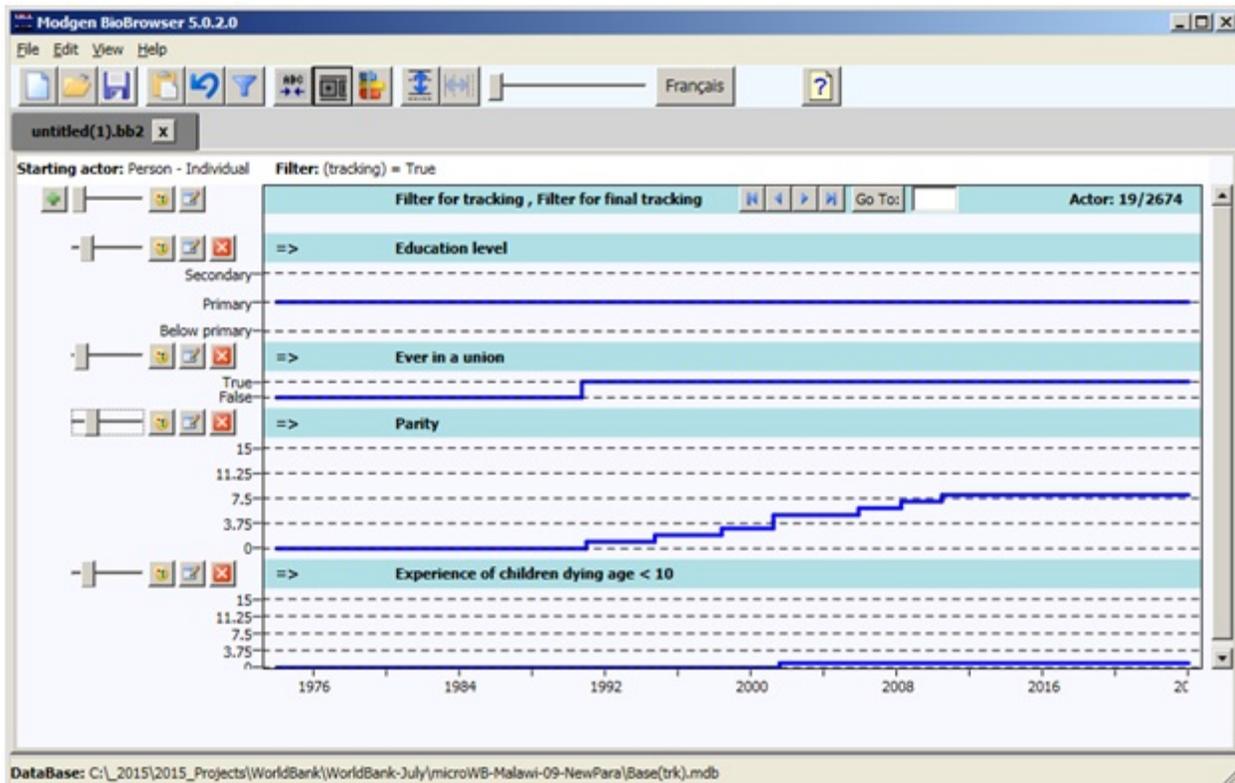
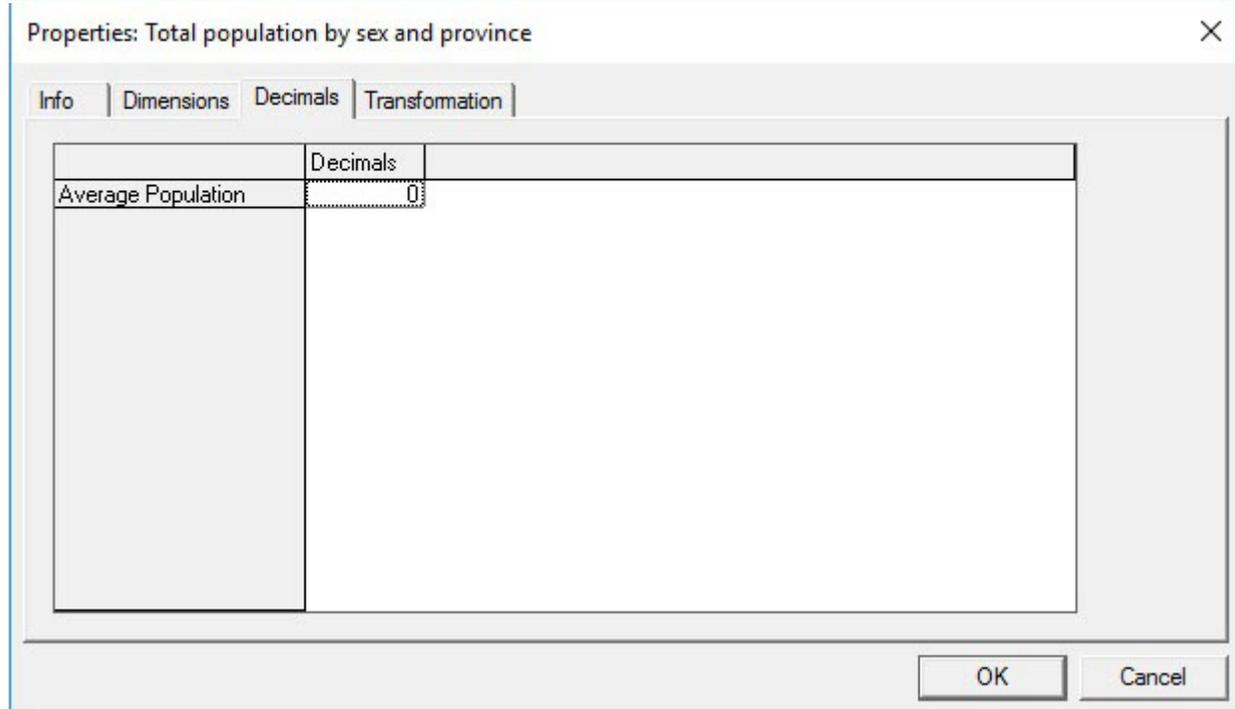
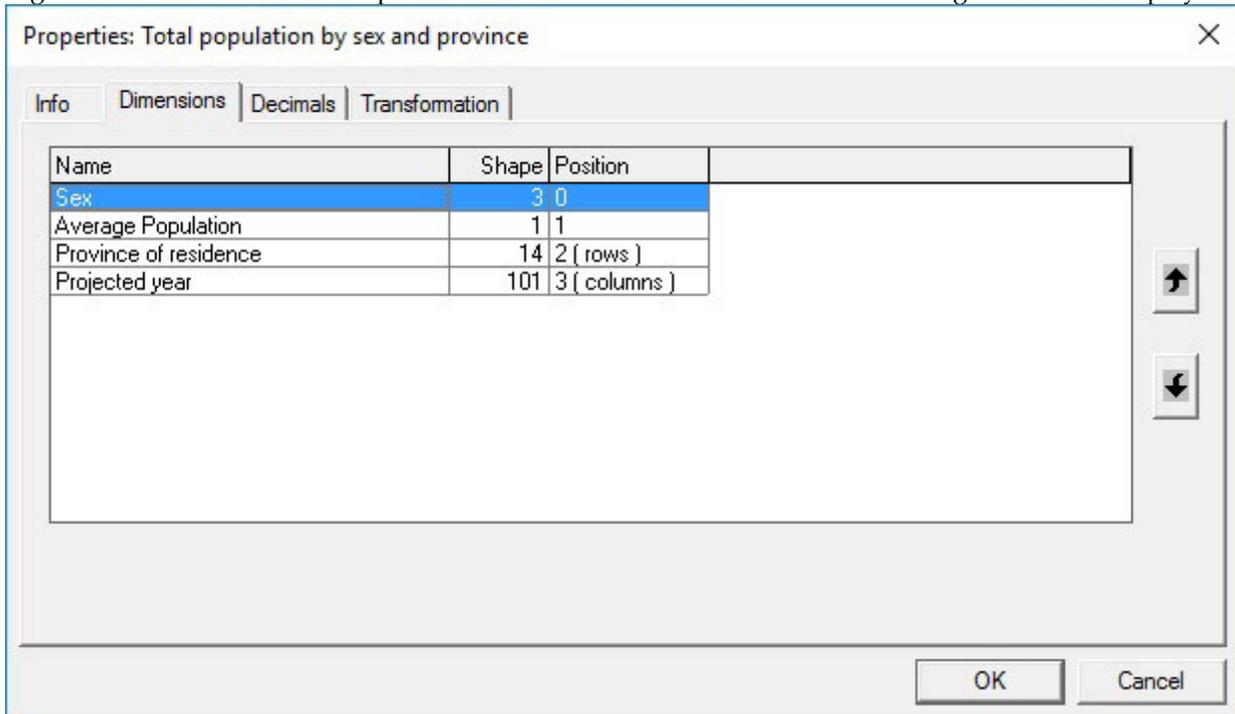


Figure 5-3: BioBrowser

5.1.4. Output Table Views

Right-click tables and select Properties to access additional information and change the table's display.



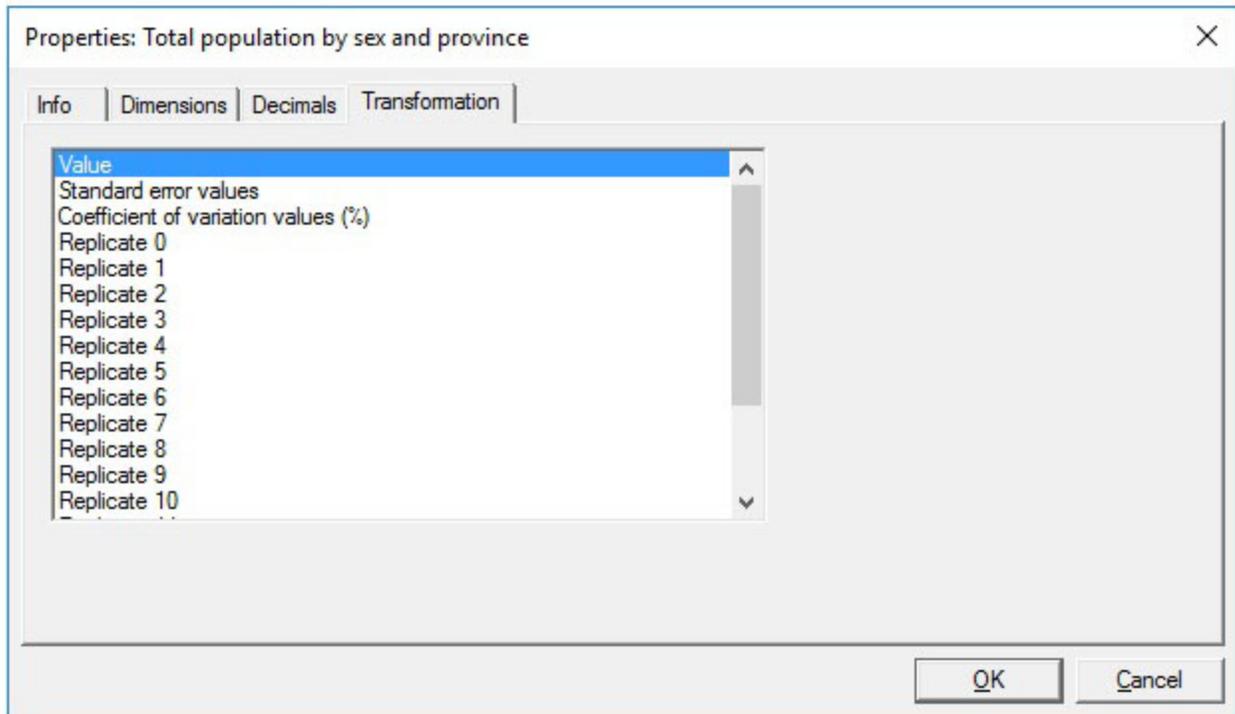


Figure 5-4: Table Properties

- Info tab: contains information on the internal table name, its label, and, if available, a note
- Dimensions tab: allows users to change the order of dimension in which a table is displayed
- Decimals: allows users to change the number of decimal spaces
- Transformation: selects alternative types of values for display. The default is Value, which is the simulation results. If several replicates are run, results of individual runs can be displayed, or, most importantly, distributional information accessed

5.1.5. Exporting Simulation Results

There are three ways to export table results for further use and analysis (e.g., in Excel):

- The content of individual tables or table cells can be directly copied and pasted, either by marking cells or by right-clicking on a table and selecting copy from the Context menu. In this case, table values are copied without row, column, or table labels.
- Right-clicking on the table also enables the Copy Special option. Here, the current table view or all dimensions of the table together can be copied, including all dimension and table labels.
- The Scenario/Export menu lets users choose tables to be exported together into an Excel Workbook. Dialog selects the display format for tables (including pivot table format) and chooses a file name.

5.1.6. Further documentation

The user interface is fully documented within the application. Users can use the help menu to access a detailed hyperlinked help option, which covers all capabilities relevant to model users, such as editing parameters, creating scenarios, running the model, and viewing and exporting model results.

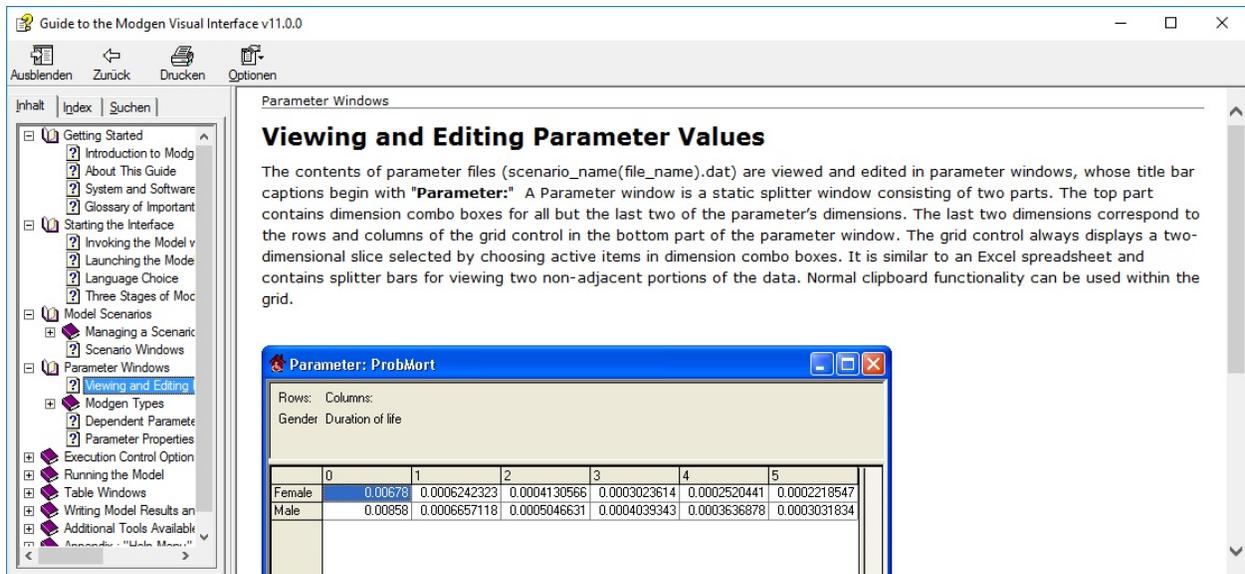


Figure 5-5: The Help System Documenting the User Interface

Similar to the model’s user interface, the model itself is fully documented within the application. Users can access encyclopedic documentation from the help menu, including descriptions of the modules, parameters, model actors, and all table output.



Figure 5-6: The Model Documentation System

5.2. The Developer’s Perspective: Key Programming Concepts

This chapter gives some flavor on Modgen programming, and an extensive discussion of most Modgen concepts is provided in Chapter 7, with a detailed guide to implement the population projection model in 18 steps. A detailed technical developer’s guide is also available for the Modgen language; it is integrated in the Modgen’s help option and available as a pdf at <http://www.statcan.gc.ca/microsimulation>.

Modgen requires and integrates into the Microsoft Visual Studio C++ package, thus making use of one of the most popular programming interfaces and its powerful debugging tools. Modgen translates Modgen

code into C++ code, which is then compiled into a C++ application, combining the strengths of the generic C++ language with specialized micro-simulation language concepts and functions. A model developer does not need to have advanced programming skills to use Modgen, because Modgen hides underlying mechanisms like event queuing and automatically creates a stand-alone model-executable program with a complete visual interface and detailed documentation. Developers can therefore concentrate on model-specific code: the declaration of parameters, simulated actors, states, events, and table output.

5.2.1. Parameter Dimensions and Tables

All parameters in Modgen are organized in tables, which can be as simple as a single on/off checkbox or a multidimensional table. Data types can be logical, integer, or real numbers. To define parameter dimensions, Modgen has three key concepts: classifications (for categorical variables), partitions (to split a continuous variable or time into pieces), and ranges (a set of integer values). Parameters are declared in a `parameters {};` block.

```

classification SEX //EN Sex
{
    FEMALE, //EN Female
    MALE //EN Male
};
partition AGE15 { 15, 30, 45, 60, 75 }; //EN 15 year age groups
range PROJ_YEARS { 2015, 2060 }; //EN Projected years
range

parameters
{
    double EmigrationRate[SEX][AGE15][PROJ_YEARS]; //EN Emigration Rate
};

```

Note that the code comments, which is the text introduced with `//EN` (where “EN” is defined as English language; Modgen supports multi-lingual models), are used to automatically label the resulting tables in the user interface. Use the code above to create a parameter table to display on the user interface and make accessible in the program.

	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	
(min,15]	0.0100	0.0100	0.0100	0.0100	0.0100	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0102	0.0102	0.0102	0.0102	0.0102	0.0102	0.0102	0.0102	0.0102
(15,30]	0.0200	0.0200	0.0200	0.0201	0.0201	0.0201	0.0201	0.0201	0.0202	0.0202	0.0202	0.0202	0.0202	0.0203	0.0203	0.0203	0.0203	0.0203	0.0204	0.0204	0.0204	0.0204	0.0204	0.0204
(30,45]	0.0300	0.0300	0.0301	0.0301	0.0301	0.0302	0.0302	0.0302	0.0302	0.0303	0.0303	0.0303	0.0304	0.0304	0.0304	0.0305	0.0305	0.0305	0.0305	0.0306	0.0306	0.0306	0.0306	0.0306
(45,60]	0.0200	0.0200	0.0200	0.0201	0.0201	0.0201	0.0201	0.0201	0.0202	0.0202	0.0202	0.0202	0.0202	0.0203	0.0203	0.0203	0.0203	0.0203	0.0204	0.0204	0.0204	0.0204	0.0204	0.0204
(60,75]	0.0100	0.0100	0.0100	0.0100	0.0100	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0101	0.0102	0.0102	0.0102	0.0102	0.0102	0.0102	0.0102	0.0102	0.0102
(75,max]	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0050	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051

Figure 5-7: The Parameter Coded Above

The parameter values are stored in `.dat` text files, usually in groups or all parameters together in one or several file(s). The syntax is identical to the programming code, except that values are provided:

```
double EmigrationRate[SEX][AGE15][PROJ_YEARS] = { value1, value2, ...};
```

5.2.2. Actors, states, functions and event declarations

Actor is an entity whose life is simulated in a Modgen model. There can be different types of actors in a model, and Persons is typically the most important. Actors are characterized by state variables as well as

the functions and events that change the states. Some states can be numerical (e.g., age, income) and others are categorical (e.g., gender, marital status). 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.

Modgen supports three types of state variables:

- “Simple” states those that are updated in events
- “Derived” states those that depend on other states and update themselves, e.g., a formula in an Excel table cell
- “Self-scheduling” states those that update themselves following a pre-specified time schedule

Two state variables, time and age, are supplied and maintained automatically. States and events belong to a specific actor and are declared in an actor block:

```
actor Person
{
    SEX          sex = { FEMALE };           //EN Sex
    logical      resident = { TRUE };        //EN Resident

    int          age_years = self_scheduling_int(age); //EN Age in full years
    int          age_index = split(age_years, AGE15); //EN Age group index
    int          simulated_year = self_scheduling_int(time); //EN Calendar year

    event        timeEmigrationEvent, EmigrationEvent; //EN Emigration Event
};
```

In the previous example, we define:

- Sex and resident as simple states, initialized with a value in {} brackets
- Age_years as a self-scheduling state, creating its own schedule to be updated exactly at the right moment independent of other events in the model
- Age_index is a derived state, when age_years changes, it is updated when a new age group limit is achieved
- An event “EmigrationEvent”

5.2.3. Event Implementation

Each event consists of two functions, one returning the time of an event, the second accessed if the event happens. Typically, the time of an event is based on piece-wise constant hazard models, with the waiting time assumed to be exponentially distributed for a given hazard rate, which is assumed to be stable over specific pieces or episodes of time. For example, it is assumed that the emigration hazards will stay constant for a given age range and calendar year. When the age group or calendar year changes, Modgen automatically generates a new waiting time based on the new applicable rate. For a given hazard rate, a random waiting time can be calculated as $-\ln(\text{RandUniform})/\text{hazard}$.

```
TIME Person::timeEmigrationEvent()
{
    double dHazard = EmigrationRate[sex][age_index][RANGE_POS(PROJ_YEARS,
calendar_year)];

    if ( resident && WITHIN(PROJ_YEARS, calendar_year) && dHazard > 0.0)
    {
```

```

        return WAIT(-log(RandUniform(29)) / dHazard );
    }
    else return TIME_INFINITE;
}

void Person::EmigrationEvent()
{
    resident = FALSE;
}

```

For persons at risk of emigration (i.e., residents within the simulated timeframe and positive hazard rate), a random waiting time to emigration is calculated based on the given rates. Note that Modgen automatically handles an event queue and automatically updates waiting times whenever a state affecting the waiting time changes. The second function, `EmigrationEvent()`, implements emigration itself: the event sets the simple state “resident” to FALSE.

5.2.4. Table output

Modgen contains a powerful tabling language, and even complex tables usually require only a few lines of code. Tables belong to an actor and consist of a name, a label, and one or more table expressions, e.g., formulas for the calculation of output values. Tables can be divided by an unlimited number of dimensions (e.g., output by age and sex) and can contain a filter which selects who should be included in the table and/or when the table should be created.

For example, the following table calculates the simulated emigration hazards by age group, sex, and calendar year. For sufficiently large simulated populations, the simulated rates should come close to the parameter.

The table filter (second line, within [] brackets) selects the timespan for which the table is produced. The table dimensions are projected calendar years, age group, and sex, as in the parameter table that drives the model. Age groups can be built automatically in the table for a given partition using `split()`. The sign after a table dimension indicates that the table should also calculate totals over all categories of a variable.

```

table Person SimulatedEmigrationRates //EN Simulated
Emigration Rates
[ WITHIN(PROJ_YEARS, calendar_year)]
{
    sex+ *
    {
        transitions(resident, TRUE, FALSE) / duration(resident, TRUE) //EN Emigration
Rate decimals=4
    }
    * split(age_years, AGE15) //EN Age group
    * proj_years
};

```

Modgen has a long list of functions used in tables. The functions used here are `transitions()`, which count the number of specific transitions specified, and `duration()`, which calculate the time spent in a certain state. With these functions, rates can easily be calculated as the number of events divided by the time at risk.

	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039
(min,15]	0.0098	0.0101	0.0092	0.0104	0.0097	0.0118	0.0100	0.0088	0.0102	0.0105	0.0088	0.0091	0.0102	0.0094	0.0099	0.0098	0.0099	0.0099	0.0111	0.0099	0.0106	0.0108	0.0108	0.0116	0.010
(15,30]	0.0176	0.0210	0.0196	0.0209	0.0204	0.0190	0.0193	0.0195	0.0201	0.0218	0.0197	0.0212	0.0210	0.0201	0.0201	0.0205	0.0202	0.0214	0.0211	0.0217	0.0220	0.0216	0.0209	0.0205	0.022
(30,45]	0.0316	0.0309	0.0279	0.0305	0.0311	0.0289	0.0317	0.0296	0.0292	0.0297	0.0269	0.0299	0.0343	0.0336	0.0289	0.0328	0.0302	0.0294	0.0282	0.0330	0.0308	0.0314	0.0298	0.0305	0.031
(45,60]	0.0202	0.0172	0.0222	0.0210	0.0222	0.0230	0.0227	0.0187	0.0221	0.0185	0.0208	0.0245	0.0210	0.0197	0.0190	0.0204	0.0236	0.0212	0.0210	0.0237	0.0202	0.0227	0.0180	0.0206	0.018
(60,75]	0.0073	0.0112	0.0085	0.0079	0.0090	0.0120	0.0113	0.0094	0.0095	0.0129	0.0088	0.0081	0.0121	0.0127	0.0135	0.0096	0.0099	0.0084	0.0076	0.0103	0.0083	0.0070	0.0136	0.0120	0.012
(75,max]	0.0056	0.0116	0.0039	0.0171	0.0040	0	0.0077	0.0057	0.0054	0.0071	0.0034	0.0068	0.0032	0.0044	0.0103	0.0071	0.0068	0.0052	0.0087	0.0115	0	0.0060	0.0058	0.0055	0.001

Figure 5-8: An Output Table Displaying Simulated Rates

5.2.5. Model documentation

Modgen automatically produces a hyperlinked help file containing model documentation based on the model code as well as labels and notes within the code. Variable labels and other information within the code are also used for producing a fully labeled user interface, including labeled output tables (e.g., table name, variable names, and description of variable dimensions).

5.2.6. Programming Wizards and Model Templates

In addition to the ample functionality provided automatically, with the underlying code hidden from the developer, Modgen also provides code wizards to generate the essential code of new models, add micro-data input modules, and create new “empty” modules. Code produced by the wizards is visible, but usually does not require any changes by developers, or just minor adaptations, like the specification of variable names in the case of micro-data input files.

Alternatively, developers can start from existing models. The step-by-step creation of the population projection model supports this approach, as the first steps are very generic and modelers can depart from such earlier steps when building models. In particular, the code was created in a way that makes it easy to switch between the two main model types, case-based and time-based (the latter allows for interactions between actors, while former simulate one case at a time), and between models starting from distributional tables and those using a starting population file. Models can be easily converted between all four combinations of types in all steps up to 9; step 10 demonstrates the conversion from a model that starts from a distributional table to a model that reads a starting population file, in our case, a file produced by the model itself at Step 9.

Chapter 6. Statistical Measures, Concepts, and Methods Common in Micro-simulation

6.1. Starting a Simulation and Keeping it Going: Population, States, Events

Dynamic micro-simulation creates a realistic depiction of a population and its changes over time by simulating a large sample of individuals, i.e., actors, who have the same distribution of individual characteristics as the real population. To start a simulation, a starting population file of micro-data records representing persons by a set of characteristics, i.e., states, and optional links to other actors, such as family or household members, is usually read in. During the simulation, as actors age, some of their characteristics and links change, and eventually they die. New actors are created and added at birth or after immigration, while others may leave the population due to emigration. Classic population projection models concentrate on three demographic processes: fertility, migration, and mortality. The model developed in this report adds first union formation and educational attainments to this list.

Processes correspond with events that can happen at a single moment of time and change individual states:

- Birth events increase the parity of the mother. At this event also the date is recorded for the modeling of following births, and a new actor—the baby is created, inheriting some characteristics of the mother, like the province of residence. A permanent link between child and mother is created, allowing the child to access her mother’s characteristics. In our model, child mortality depends on mother’s age and education at birth, and the child’s future educational attainments are influenced by mother’s education.
- Migration events change the province of residence of individuals. At immigration, new actors are created and their characteristics have to be initialized.
- At death, the actor and links to other actors are removed.
- At first union formation, an indicator is set and the time of the event is recorded. In more detailed models, partners are searched in the population and couples are linked.
- We model two primary education events: entering and graduating primary school.

To realistically model events, we need measures of their occurrence in reality. These can be measures collected for a current period, such as a death register recording deaths by basic characteristics like age and sex. In our case, the most important data source is a population census, which collects both the current characteristics of individuals and households and information on some important recent events like the number of births given in the past 12 months, origin and timing of the most recent migration event, and the date at first marriage. A third data source is surveys, which typically are more detailed, but based only on a sample of the population. In our case, valuable survey information is birth histories of

women used for the modeling of the timing of births (birth intervals). In addition, we use history data on children’s survival—respectively, the date of their deaths reported by mothers—as the base for child mortality modeling.

6.2. Measures of the likelihood of events: probabilities and rates

The two most important measures for the likelihood of events calculated from data are probabilities and rates.

- Probability describes the likelihood that an event will occur for a single individual in a given time period and ranges from 0 to 1. It is calculated as number of events that occurred in a time period divided by the number of people followed for that time period. Note that such a measure is possible only if the event can happen only once per individual and time unit. If repeated events are possible, probabilities can still be calculated if at least one event was experienced, or probability distributions can be calculated if 0, 1, 2, or more events were experienced.
- Rate is an instantaneous measure. While also based on a count of events, the measure includes time—specifically “time at risk”— in the denominator. As an example, if the probability of 50-year-old people dying within a year after their 50th birthday is 1 percent, this means that one person out of 100 alive on her 50th birthday will not experience her 51st birthday. The 1 percent is calculated by dividing the number of deaths by the original number of people. In contrast to probability, for rate, we divide the number of events by the time people were at risk. Those who died during the year were not at a greater risk of death after the date of death. If from a population of 100 people one person died, depending on when the death occurred, the calculated rate would be between 0.01 (1/100 if the death occurred at the end of the period) and 0.0101 (1/99 if the death occurred at the beginning of the period). Rates and probabilities are very close for unlikely events and therefore often confused. In contrast, if 90 percent of the 100 people die in an accident, depending on the time of the accident, the measured rate would be between 0.9 (90/100) and 9 (90/10). This huge span in measures will of course disappear if we have a big sample, if events are spread over time, and if we can assume a constant risk over this time unit. If risks change over age or time, we typically measure them for single years of age or calendar years, which are usually small enough intervals for the assumption to approximately hold.

Under the assumption of a constant risk over time span t , rates are mathematically linked to corresponding probabilities of survival, i.e., the probability that the event does not happen to a person over this time t :

$$S = \exp(-r*t)$$

If an event can happen only once, its probability of occurrence is $1-S$. If we know the probability of survival, under the assumption of a constant risk over time t , we can calculate the corresponding rate as:

$$r = -1/t * \ln(S)$$

The likelihood an event will occur typically depends on specific circumstances. For example, mortality changes with age, but may also be impacted by other factors. For instance, when modeling child mortality, we also account for mother’s characteristics. Accordingly, measures of likelihood have to be available in a simulation for each combination of influencing factors. This can be achieved in two ways. First, if there are only few combinations and the sample size is large enough, we can calculate measures for each of all possible combinations of circumstances. The second approach is to find a formula that, when based on circumstances, can calculate the likelihood of an event. A very popular class of such formulas are proportional models. In the case of rates, proportional hazard models divide risk into baseline risk (e.g., mortality by age) and relative risk, assuming that specific circumstances (e.g., health status) modify this baseline proportionally over all ages. One of the convenient characteristics of rates is that they can be

directly multiplied by relative risk factors. For example, a 100 percent increase in risk can be directly expressed as a multiplication of the initial risk by the factor 2. This does not work for probabilities. For example, when doubling a probability, the resulting number has no interpretation of a proportional probability; in fact, the calculation may even lead to a value above 1. To overcome this problem and allow estimation of proportional models that can be used to derive probabilities, transformations are used. A typical model of this class is logistic regression, which instead of estimating probabilities directly, estimates the log odds of an event. Odds are a transformation of probabilities typically used in gambling: a 75 percent chance to win can be expressed as a 75:25 chance, respectively, with the odds measuring 3.

$$\text{odds} = p / (1 - p)$$

Like rates, odds can be multiplied by proportional factors, which can be estimated in regression models.

6.3. From Measures to a Simulation

Corresponding to the two basic measures for the likelihood of events—probabilities and rates—we can distinguish two types of dynamic simulation models: discrete time and continuous time.

Models based on probabilities are discrete time models, i.e., models in which states are updated in fixed time steps, such as each year. In the simulation, if an event that happens to an individual is a random process based on the known probability, we draw a random number between 0 and 1 and decide that an event has happened if the random number is smaller than the probability. Probabilities contain no information on when an event happens within a time period; the same is the case for discrete time models that just model the before and after. Therefore, if more than one state changes in such models, there is no information available as to which event happened first. Also, we do not know the history of a person within this period, so a person now living in the capital who lived there a year ago may have spent most of the year somewhere else and moved several times. This is not a big problem for rare events for which multiple occurrences within a single period are very unlikely. Also, sometimes data do not allow more accurate descriptions of processes, as a survey could collect just the information now and a year ago. A comparable situation is the use of panel data, which is linked cross-sectional data that collect states as snapshots at various points in time.

Models based on rates are continuous time competing risks models in which events can happen at any moment in time. Instead of moving time in single steps of fixed length, such as years, and updating all states at once, time advances with whatever event happens first. At the start of the simulation, the rates for each event are calculated and this information is used to calculate waiting times for all possible events. Assuming a constant risk over time, the waiting time to an event follows an exponential distribution and a random waiting time based on a given rate can be easily calculated using a simple formula.

$$\text{random waiting time} = -\ln(\text{RandomNumber}) / r$$

The expected waiting time of an exponentially distributed random variable is just the inverse of its rate:

$$\begin{aligned} \text{expected waiting time} &= 1 / r \\ \text{median waiting time} &= -\ln(0.5) / r = 0.69 / r \end{aligned}$$

Events compete, meaning that the event scheduled to happen first is the one that will move time to the time point at which it happens. The occurrence of an event typically impacts other events. For example, for women in reproductive age entering a first marriage, fertility risks typically increase. At the same time, the woman is not at risk for entering a first marriage anymore, but becomes at risk for divorce (if modeled). Accordingly, at the occurrence of each event, a new list of possible events is created and all waiting times affected by the event are updated.

The assumption of a risk being constant typically only holds for specific periods or “pieces” of time, so we talk about piece-wise constant risks. For example, even if nothing else happens, a waiting time has to be

updated if it exceeds the piece of time for which the underlying rate is valid. For example, mortality risks stemming from life tables are typically valid for a single year of age, while the waiting time to death has to be updated at each birthday.

The micro-simulation model developed in this report follows a continuous time approach and most events can happen at any point in time. Note that the approach does not limit one to using only models based on rates. For example, we model school entry and graduations once at a fixed point in time each calendar year and the decisions to enter school respectively to graduate are based on probability parameters.

6.4. Estimating and calibrating probabilities and rates

In the simplest case, rates or probabilities used in a micro-simulation model can be obtained directly from published sources. In our case, age-specific mortality and fertility rates in the base versions of the model stem from rates provided by the ONS and are identical to rates used in macro population projections. More typically, micro-simulation requires more detailed measures, accounting for their variation by individual characteristics.

When selecting statistical methods for deriving rates and probabilities used in the simulation, we mostly make use of proportional models. Such models are very useful in micro-simulation; model builders, can combine information from different data sources and model users can support easy-to-interpret scenario creation. Proportional models divide the likelihood of events to a baseline factor that applies to all, and to relative factors based on specific individual characteristics. Accordingly, scenarios can be created in which the likelihood changes for all individuals, e.g., a mortality trend lowering overall mortality over time, or only for specific groups. Empirically, relative differences in risks between population groups are often very persistent over time and it can be a reasonable assumption that these differences persist also in the future, at least in the absence of specific policy interventions.

In contrast, for policy analysis, convergence scenarios where we assume that a gap between population groups like ethnicity can be closed over time are very helpful, as they can be used to study the downstream effects of such a change. But if there are no changes, proportional models are useful as status quo scenarios where we assume no change both in the baseline and the relative factors. While such scenarios assume stability in behaviors on the individual level, aggregate outcomes will still change over time if the composition of the population changes. An important strength of micro-simulation models is this ability to distinguish composition change from behavioral change. This modeling approach can be best explained by means of the following five examples.

Example: calibrating published baseline rates for fitting aggregated trends

For mortality modeling, we use published mortality rates stemming from a standard life table by age and sex. In the simulation, we use these rates as baseline hazards by age. At the same time, we support scenarios for increases in life expectancy over time. To do so, we introduce a second parameter: life expectancy by calendar year. To reach this target life expectancy, the mortality baseline is proportionally scaled by a factor that, when multiplied to all age-specific rates, modifies the life table so that the target life expectancy is reached. In our case, this factor is found by numeric simulation automatically performed in the model; statistically, it is a relative risk that decreases over time, as life expectancy increases.

Example: baseline hazard and relative risks estimated simultaneously

When modeling child mortality, we estimate the relative risks of birth by mother's education and mother's age, together with the baseline hazards by age of the child from retrospective birth and death history data using a piece-wise constant hazard regression model. For estimating such a model, individual records are split into single records for each time span, in which all covariates are the same: in our case we get one record for each year a child is alive until age five. In contrast to this time-changing covariate (age), other covariates (mother's age group at birth, mother's education level) stay the same in each of the

created records. The only other two pieces of information needed to estimate the model are the time at risk in each time span (in our case, a year if the child survives, respectively the time at this age until the date of the survey, or the time until death if the child dies in a given year) and an indicator if the event (i.e., mortality) happened. The results of the regression are baseline hazards—death rates by age—and relative risks by mother’s education level and age group. Note that, by accounting for the time at risk, hazard regression models have no problem with “right censoring,” i.e., the fact that observations end at the moment they are collected, which is not necessarily at the end of time intervals of the model (like age in our example). Also, the exit of persons from the sample for other reasons than the modeled event (e.g., by emigration) can be easily handled by this approach. Mathematically, a piece-wise constant hazard model can be written as:

$$r_{ij} = a_j * \exp(b'_{i} X),$$

r_{ij}	hazard rate for an individual i in time interval j
a_j	baseline hazard for the time interval j
b'_{i}	(transposed) vector of individual characteristics
X	vector of relative risks

Example: combining base line hazards and relative risks stemming from different sources

One of the strengths of the proportional modeling approach is the fact that baseline hazards and relative risks can stem from different data sources; this allows combining information from robust data sets for the baseline (e.g., administrative data, census) with far more detailed survey data for estimating relative risks. We used this approach as an option in modeling child mortality, combining the overall mortality rates from the base model, but combining those with the relative risks estimated for child mortality risks. Combining information from two sources requires calibration very similar to that in the first example above: for a given population composition by the characteristics used for relative risks, the application of relative risks to the calibrated baseline risk must result in the target baseline rate. (Again, this step is automatically performed in the micro-simulation model). Cox regression is a statistical model widely used in literature for estimating relative risks.

All three of these examples were examples of rates and use the convenient feature that they can be directly multiplied. Doubling a rate means doubling the risk, which reduces by half the expected waiting time to the event. As noted above, probabilities cannot be linearly transformed as rates; also, as probabilities are limited to an interval 0 - 1, they are typically not estimated directly in regression models, as a mechanism has to be applied to limit regression results to the allowed interval. The most widely used transformation for estimating the probability of events are logits (log odds) as used in logistic regression:

$$\text{logit}(p) = \ln(p / (1-p)) = a + bX$$

As a result of the logistic regression, we get the coefficients a —the baseline—and a vector b of proportional factors from which we can calculate the probability of an event for a given vector of individual characteristics X .

$$p = \exp(a+bX) / (1 + \exp(a+bX))$$

The estimated coefficients are interpreted as log odds, or, more intuitively, odds ratios when calculating their exponential. For example, applying an odds ratio of 2 means, that the odds of an event double from 1: x to 2: x . If chances were initially 50:50, they become 100:50, i.e., 2:1, or, expressed in probabilities, increase from 50 percent to 66 percent.

Example: Logistic regression for modeling provincial differences and time trends

In our micro-simulation model, we used logistic regression models for deriving probabilities to enter and graduate from primary school, as one example. These models contain a baseline, a time trend, and proportional factors for provincial differences. Analysis has shown that provincial differences, expressed in log odds, are very persistent over time, the proportionally of the model therefore being a reasonable assumption. Accordingly, the base scenario assumes that these differences also persist in the future. Users

can easily run alternative scenarios by, for example, assuming convergence of probabilities to the best or, as presented in this report, assuming universal education is introduced immediately or phased-in over the next 10 years for all provinces.

Example: Adding additional odds-ratios to a model

Like the case for continuous time models based on hazard regression, discrete time models based on probabilities also allow adding additional relative factors that might stem from different data sources. For example, in the models for primary school enrollment and graduations, we provide an option to add odds ratios by mother's education, allowing the study of the inter-generational transmission of education. If this option is chosen, additional factors are added in a way that, by calibrating the base odds at the beginning of the simulation or a specific year selected by the user, the overall probability of entering school is not changed. From this year onward, we then freeze the new calibrated base odds. As a result, all projected changes on the aggregate level can be attributed to the changing educational composition of mothers.

6.5. Probability Distributions and Origin-destination Matrices

Probability distributions are used for models of more than two possible outcomes. An example is the distribution of destination provinces of immigrants. If the destination distribution depends on the origin, distributions can be tabled in origin-destination matrices. This approach is used for modeling internal migration, further divided by age group and sex. Like single probabilities, probability distributions and origin-destination matrices can be either directly tabulated from micro data or modeled; the choice depends on sample size and desired number of co-variates. Multinational extensions exist for the logistic regression approach presented above; alternatively, multiple outcomes can be modeled by decision trees applying binomial models.

6.6. Micro-simulation Implementation

Once the required probabilities or rates are obtained and all regression coefficients are estimated, this information can be used to parameterize the micro-simulation model. Parameter tables can directly contain rates or probabilities by any number of dimensions (e.g., age, sex, region) or contain the regression coefficients, leaving the calculation of the used hazards or probabilities to the model. Model parameters are used in various ways:

- Some parameters can be used directly. For example, in the base version of education, whether to admit people of school entry age into school is directly based on probabilities by calendar year, sex, and province contained in a three-dimensional parameter table. While the probabilities stem from a logistic regression of log odds, regression results were transformed outside the micro-simulation model into probabilities, making the parameter table more intuitive.
- Some parameters require further calculations performed within the micro-simulation whenever they are needed. This is the case for parameter tables containing regression coefficients. For given individual characteristics, the regression formula is evaluated using the parameters. For example, child mortality is implemented using two parameters, an age baseline and relative risks. Based on this information, individual risks can be calculated by multiplication and updated at each birthday.
- Some parameters are created within the model based on other parameters before the start of the simulation. For example, a "model-generated" parameter table of mortality hazards by age, sex, and calendar year is created before the simulation by scaling the values from a standard life table parameter to reach a target life expectancy for each year that stems from another parameter.
- Some parameters are used indirectly as alignment targets. They are not used to schedule individual events but to set target values in the simulation. For example, in one model we use the age-specific

fertility rates of the model's base version to determine the target number of births matching existing macro projections, but use a more detailed model, including parity, education, and time since last birth, to distribute births to the most likely mothers based on relative differences in birth risks beyond age alone. In this case, the individual waiting times based on the refined model are used for ranking potential mothers.

During simulation, probability and rates are used to create individual events or decisions. Processes based on probabilities or rates are stochastic processes, meaning that they contain a random element that determines if or when an event happens to an individual. When simulating a large enough sample of individuals, the proportion of people experiencing an event of a given probability will come close to the overall probability, and the simulated number of events will come close to the expected value based on given rates. For individuals, transitions are binary.

In the case of rates, waiting times are calculated and put into an event queue, and the first event to happen censors other events in the queue and causes an update of all other processes affected, as described previously. As for probabilities, the individual waiting times are stochastic, thus dependent on a random number in combination with the rate. If the sample is large enough, the average waiting time in the simulation will come close to its expected value, while individual waiting times are exponentially distributed.

In the case of probabilities, the model has to create regular (e.g., yearly) events when a process or processes are updated. For example, school entry can be scheduled on a specific day in a year. On this day, all eligible persons decide if they enter school or not based on given probabilities. Technically, this is done by drawing a uniform distributed random number between 0 and 1 and comparing the result with the probability: the event happens if the random number is below the probability. Also probability distributions and origin-destination matrices can be used directly in combination with a random number to determine the outcome of a choice, like the destination province of a migrant.

Chapter 7. A Step-by-Step Guide

This is complete guide for building DYNAMIS-POP-MRT starting from a simple model template and adding functionality at each step. For each step, the following information is provided:

- An overview what is added at this step
- New Modegn/openM++ programming concepts introduced at this step
- A step-by-step guide for what to add to complete this step, starting from the previous step

Steps 1 - 9 replicate DemProj, a typical macro population projection model, and are implemented as a so-called case-based model (i.e., a model with no interaction between simulated persons) and a starting population created by sampling from a distributional table of persons by age, sex, and province.

- 7.1. Step 1: Model Templates
- 7.2. Step 2: Creating a Population
- 7.3. Step 3: Population Scaling
- 7.4. Step 4: Fertility
- 7.5. Step 5: Refining Mortality
- 7.6. Step 6: Migration
- 7.7. Step 7: Emigration
- 7.8. Step 8: Immigration
- 7.9. Step 9: Micro-Data Output

Step 10 converts the starting population type to a micro-data file. Up to this point, models of each step can also be converted to time-based models, which can be used as templates for other applications. This is illustrated in Step 11, which creates a time-based starting template for the next phase of model development going beyond the replication of a macro model.

- 7.10. Step 10: Micro-Data Input
- 7.11. Step 11: Conversion to a Time-Based Model

Steps 12+ build an interacting population micro-simulation model step-wise, introducing new behaviors like first union formation, education, and refined models for demographic behaviors such as fertility, (infant) mortality, and migration by education.

- 7.12. Step 12: Sampling from a detailed starting population
- 7.13. Step 13: Primary Education

- 7.14. Step 14: First Union Formation
- 7.15. Step 15: Fertility refined
- 7.16. Step 16: Infant mortality
- 7.17. Step 17: Migration refined
- 7.18. Step 18: Extending table output

7.1. Step 1: Model Templates

7.1.1. Overview

Model Templates

DYNAMIS-POP-MRT model development starts from a generic model template for continuous-time models. The model creates a cohort of people all born at age 0 and subject to age-specific mortality. Mortality is the only stochastic event at this step. Accordingly, the model has only one parameter: mortality rates by age. The population size is part of the scenario settings and can be chosen by the user, too. The model output consists of a set of tables, which have mostly demonstrative use. They contain demographic measures like life expectancy, and, for model validation, age-specific mortality rates calculated from the simulated cohort of people.

The template contains the following mpp files, which contain the following model codes:

- DYNAMIS-POP-MRT.mpp	The simulation engine which handles the simulation run
- ModelSettings.mpp	A file with settings specific to the model version
- PersonCore.mpp	A core file of the actor Person
- Mortality.mpp	Mortality module
- Tables.mpp	The module containing all output tables

The template also contains the data file:

- Base(PersonCore).dat	File containing all parameters common to all versions at this step
- Base(ModelSettings).dat	File containing all version-specific parameters of this modeling step

Model Versions

This step-by step modeling guide and documentation cover four model types and allow conversion between types at any step until Step 8, which completes the model's base version, the replication of a macro model. Besides the introduction of type-specific concepts, this supports the use of the model steps as templates for other models.

- **Case-based versus time-based models:** Time-based models allow for interactions between all persons. While no interactions are required in the base version of DYNAMIS-POP-MRT, they are important in the following model extensions. The DYNAMIS-POP-MRT.mpp file, i.e., the simulation engine, is the only file specific to this model distinction; all other modules except ModelSettings are generic. All case-based models can be converted to time-based models by replacing the simulation engine and modifying a few settings in ModelSettings.mpp.
- **Sampled versus file-based populations:** The base version of DYNAMIS-POP-MRT projects a population by age, sex, and province, which is the only information required for the starting point of the simulation. This information can be contained in a distributional table from which the model samples; alternatively, information can be contained in a micro-data file. The first approach works

only for a very small number of variables and all variables must be categorical, a limitation that can be overcome by representing the population in a file. This will be important when extending the model beyond its base version. While building the base version, the distributional table can be replaced by a file at any modeling step, starting with Step 1, by replacing the starting population file and changing a few settings in `ModelSettings.mpp`. This will be documented in the next step of model building.

The most efficient model type for the base version of DYNAMIS-POP-MRT is a case-based model that samples characteristics from a table. Time-based models starting from a micro-data file are the typical approach for most micro-simulation models.

7.1.2. Concepts

Overview

The code of this model template gives a first overview of main Modgen programming concepts. At this step we cover the following topics:

- Modular code organization
- Code organization within modules
- Modgen types
- Creating parameter tables
- States
- Events
- Random numbers
- Functions `Start()` and `Finish()`

Code organization in modules

Modgen model code is organized into `.mpp` files. While in theory, all code could be put into the `.mpp` file with the name of the application (in our case `DYNAMIS-POP-MRT.mpp`), a modular organization is advised. Typically, the following files can be found:

- **The simulation engine: `DYNAMIS-POP-MRT.mpp`.** This file contains core simulation functions and definitions. When starting from a template or using the Modgen wizard, developers typically do not have to modify or understand this file code. The file contains the definition of the model type; in our case, it's a case-based continuous-time model. The second part consists of two functions that handle the simulation, one to process a single case, the second to loop through all cases.
- **Actor Core Files:** Each actor typically has a core file. In our case, we have a single actor type `Person`, thus one core file `PersonCore.mpp`. Core modules contain the basic information that defines the corresponding actor, and the functions `Start()` and `Finish()` for initializing actor states at creation and deleting the actor at death. This file is also a good place for insert code that does not belong to specific behaviors or is used by various other modules.
- **Specific modules:** These are files for specific behaviors or life course domains (like in our case `Mortality.mpp`) or functionality. Developers typically spend the most time on this module type. Also, building up a model typically consists in adding modules of this type, as will be the case here, when we add `Fertility.mpp`, `Migration.mpp`, etc.
- **Table module(s):** This module(s) contain the tables of the model. Tables can be placed anywhere in the code, and/or spread to various files. In simple models, putting all tables together, e.g., into `Tables.mpp`, is a good way to organize code.

Code organization within modules

Most modules follow this organization:

- Documentation
- Type definitions
- Parameters
- Definitions of actor states, functions, and events
- Implementation of functions and events

Documentation:

Labels and notes for modules (as well as any symbols used in the program) are used for the automatically generated model help file for users. Good documentation, therefore, is not only a best practice for model development, but also creates detailed documentation for the user.

- A **label**: a one-line description of the module
- A **note**: a more detailed description of the module

Example (from Mortality.mpp):

```
//LABEL(Mortality, EN) Mortality Module

/* NOTE(Mortality, EN)

   This module implements a simple model of mortality.
   People can die at any moment of time based on age-specific mortality rates.
   The only state of this module is 'alive' which is set to 'false' at death.
   Also at death, the Modgen function Finish() is called which clears up memory
   space.
*/
```

Placed above or in the same line of a newly introduced symbol, notes can also be written as:

```
//EN The text of the note
```

You encounter such notes throughout the code. These notes are introduced by a language code (EN for English in our case). Modgen also supports multilingual applications, in which these labels are translated into (an)other language(s). Besides code documentation, these notes are used in the user interface.

Type definitions

Modgen types are typically used as types of states, and dimensions for parameter and output tables. This step introduces the type **range**.

Example (from PersonCore.mpp): AGE_RANGE defines a range of of valid integer numbers for age, allowing a maximum age of 100 years.

```
range AGE_RANGE { 0, 100 }; //EN Age range
```

Other important Modgen types are **classification** for categorical variables and **partition** to divide continuous variables like time or age into intervals.

Examples (introduced at later steps):

```
classification SEX //EN Sex
{
```

```

    FEMALE,          //EN Female
    MALE             //EN Male
};

partition AGE5_PART { 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60 }; //EN Age
Groups

```

Parameters

Parameters have a type and can have any number of dimensions. Each parameter corresponds to a table in the user interface. Tables can be as simple as a single on/off checked box (the type of such a parameter being “logical”) or consist of multidimensional tables, like tables by age, sex, region, etc.

```

The most important Modgen types

- logical      true/false 0/1
- integer      integer numbers
- double       floating point numbers
- TIME         in continuous time models equivalent with double
- SEX          a Modgen classification declared in the code
- AGE_RANGE    a Modgen range declared in the code

```

At this step, we define a single parameter table MortalityTable for mortality, contained in the Mortality.mpp module. The parameter is of type double (a precise floating point number) and has the dimension AGE_RANGE.

Parameters are organized in **parameters** code blocks, which can be placed in multiple files for introducing parameters in the modules in which they are used.

Example (from Mortality.mpp):

```

parameters
{
    double MortalityTable[AGE_RANGE];          //EN Mortality by age
};

```

Parameters can be grouped into **parameter_group** for to better organize the user interface. This is optional and the same parameter can be placed into multiple groups, allowing for example a group of “most important” parameters. Groups can also be nested.

Example (from Mortality.mpp):

```

parameter_group PG01_Mortality                //EN Mortality Parameters
{
    MortalityTable
};

```

Declarations of actor states, functions, and events

Definitions of actor states are placed into **actor ActorName {};** code blocks, which can be placed into multiple files for declaring states and events in the module they belong to.

Example (from Mortality.mpp):

```

actor Person
{
    // States

    logical alive = { TRUE };                //EN Alive
}

```

```

// Events

event  timeMortalityEvent, MortalityEvent;    //EN Mortality event
};

```

States are the variables that describe an actor. Like parameters, states have a type. For example, the state “alive” is of type logical, and initialized with { TRUE }. Note that all initializations with values have to be put into curly brackets {}. Such states are called **simple states**; the values of simple states are changed in Events. States can also be initialized by formulas containing other states, called **derived states**, in which case (like Formulas in Excel cells) the states are updated automatically. If derived states contain a “self-scheduling” function (see below), they create their own update events and are called **self-scheduling states**.

Two states are automatically created and maintained by Modgen: **age** and **time**. These two states can be set only within the Start() function (see below).

Currently the model has only two other states: the time at birth and integer_age:

```

actor Person
{
    // States
    TIME      time_at_birth = { TIME_INFINITE };           //EN Time
    at birth
    AGE_RANGE integer_age = COERCE(AGE_RANGE, self_scheduling_int(age)); //EN Age
    in full years
    ...
}

```

The **time_at_birth** is recorded at the start of the actors’ life (currently 0 for all actors) and initialized with the Modgen constant **TIME_INFINITE**—a very distant future never to happen.

The state **integer_age** is frequently used as table index (e.g., for getting the current mortality) and as table dimension. It is a self-scheduling state of type AGE_RANGE (declared above) using two Modgen functions.

- COERCE() forces an integer number into a range.
- self_scheduling_int(age) creates an integer value based on age, automatically updating itself at each birthday. This is a very powerful Modgen concept, sparing us the manual programming of a birthday event for updating the integer_age variable.

Events: At this step, the only ‘manually programmed’ event is MortalityEvent, which, at death, changes the state alive from TRUE to FALSE. Events have two functions: a time function (when does the event happen?) and the function called when the event happens.

Implementation of events

After being declared in an actor code block, events have to be implemented. The time function returns a value of type TIME and is thus of type TIME. The event function does not return a value and is thus—this is C++ notation—void.

The syntax of C++ functions is:

```

ReturnValueType ObjectName::FunctionName(ParametersIfAny)
{
    code;
    return ReturnValue; (if function is not of type 'void')
}

```

Example (from Mortality.mpp):

```

TIME Person::timeMortalityEvent()
{
    TIME dEventTime = TIME_INFINITE;

    // check if a person is at risk
    if ( MortalityTable[integer_age] > 0.0 )
    {
        // determine the event time
        // the formula [ -log(rand) / hazard ] calculates an exponentially
distributed waiting time
        // based on a uniform distributed random number and a given hazard rate
        dEventTime = WAIT(-log(RandUniform(1)) / MortalityTable[integer_age]);
    }
    // return the event time, if the maximum age is not reached at that point
    if (dEventTime < time_at_birth + MAX(AGE_RANGE) + 1.0) return dEventTime;
    // otherwise, return the moment, at which the maximum age is reached
    else return time_at_birth + MAX(AGE_RANGE) + 0.9999;
}

void Person::MortalityEvent()
{
    alive = FALSE;
    Finish(); // Remove the actor from the
simulation.
}

```

The time function of the example is a typical time function for scheduling events in continuous time. It first checks if a person is at risk of the event (here, if the hazard rate is not 0.0; but some events will only affect certain people, like those of specific age or in specific circumstances). If a person is at risk, a waiting time is calculated. For a given waiting time, the event can be scheduled. (The function WAIT(Waiting-Time) in the code example just adds the current time to the waiting time). Modgen time functions update themselves whenever something happens that affects the waiting time. In our example, this is the case if `integer_age` changes, as at each birthday, a new age-specific hazard rate applies. Thus, the two competing events of our model are birthdays and death. If a person survives until the next birthday, the scheduled death event is automatically removed from the event queue and a new waiting time is determined, until death wins the race.

Uniform distributed random numbers

Modgen uses the **RandUniform(n)** function for obtaining a uniform distributed random number between 0 and 1. The parameter (n) indicates the random number stream. Developers can leave the parentheses empty and type `RandUniform()`, and Modgen will automatically add a number. Modgen uses a different random number stream for each random function. This ensures that when adding a new random process, the sequence of random numbers of other processes stays untouched. This way, differences in results can be attributed to the added process only, and not to monte carlo disturbances by altering the sequence of random numbers. The root of all random number streams is set by the user in the simulation settings. Unless changed, re-running the same simulation will produce exactly the same results.

Special Functions: Start() and Finish()

The module `PersonCore.mpp` contains two functions, `Start()` and `Finish()`, which are part of each Modgen model and exist for each actor type separately. The functions can take parameters. In our example, `Start()` has two parameters, a record number and a link to a person. The latter allow inherited characteristics from mother to child. If not used, the function can be called with `NULL` as argument:

```

actor Person
{

```

```

...
void      Start(long PersNr, Person *prPerson); //EN Starts the actor
void      Finish(); //EN Finishes the actor
};

```

The **Start()** function is called when an actor is created. It is the only point in the code, where the state's time and age can be set and used to initiate states with values.

```

void Person::Start(long PersNr, Person *prPerson)
{
    // Modgen initializes all actor variables before the code in this function is
    // executed.
    // The Start() function is the only place where the Modgen states age and time
    // can be set.
    // The states age and time are then automatically maintained by Modgen

    age = 0;
    time = 0;
    time_at_birth = time;
}

```

The function **Finish()** has to be called when an actor is deleted in the death event. It is the place for "last-minute actions." In our example, the function is empty but will still perform automatic routines, like cleaning up the memory space. Typical code examples are writing records containing selected states and "memories" of an actor, or distributing inheritances to linked persons.

```

void Person::Finish()
{
    // Developers can add code here, currently no additional routines are required.
    // After the code in this function is executed, Modgen automatically removes the
    // actor from the
    // simulation and recuperates any memory used by the actor.
}

```

Tables

Modgen has a very powerful labeling language. At this point, some key functions and the basic syntax are introduced. The best way to learn about tables is by example and by doing. The basic table syntax is:

```

table ActorType TableName //EN Table Label
[ optional filters selecting specific actors or specific moments in time]
{
    OptionalTableDimensionA * //EN Dimension A label
    ...
    {
        FirstTableFormulaA, //EN First label decimals=n
        ...,
        LastTableFormula //EN Last label decimals=n
    } //EN Label for collection of formulas
    * OptionalTableDimensionX //EN Dimension X label
    ...
}

```

Example 1 (from Tables.mpp):

```

table Person DurationOfLife //EN Duration of Life
{
    {
        unit, //EN Number of persons ever entering the

```

```

simulation
  duration(), //EN The total time lived by all actors
together
  min_value_out(duration()), //EN Minimum duration of life decimals=4
  max_value_out(duration()), //EN Maximum duration of life decimals=4
  duration() / unit, //EN Life expectancy decimals=4

  //EN Life expectancy decimals=4
  value_at_transitions(alive, TRUE, FALSE, age) / transitions(alive, TRUE,
FALSE)
} //EN Demographic characteristics
};

```

This table has no filters and no dimensions, so everybody is recorded over the whole life. Note that the default number of decimals can be set as part of the value labels. Users can change the view of tables, including decimal paces, the ordering of dimensions.

Used table functions and keywords are:

- unit	a counter: everybody entering the table (cell)
- duration()	Time lived while in the table accumulated over all actors
- min_value_out(x)	Minimum value of x in the population when leaving the table
- max_value_out(x)	Maximum value of x in the population when leaving the table
- value_at_transitions(x, a, b, y)	Accumulated values of y when x changes from state a to b
- transitions(x, a, b)	Number of transitions of state a from level a to b

Example 2 (from Tables.mpp):

This table introduces a filter that triggers a specific moment of time: the time of death. Thus, actors enter this table at death, and disappear immediately after. Such tables are used to produce cross-sectional output at specific points in time or at specific events. Note that functions like duration() will not work and would just return 0 as the duration, at which the condition that holds true is of zero length.

```

table Person StatisticsAtDeath //EN Statistics at death
[ trigger_transitions(alive, TRUE, FALSE) ]
{
  {
    min_value_in(age), //EN Minimum duration of life
decimals=4
    max_value_in(age), //EN Maximum duration of life
decimals=4
    value_in(age) / unit //EN Life expectancy decimals=4
  } //EN Demographic characteristics
};

```

Used Modgen functions are:

- trigger_transitions(x, a, b)	True at the moment the value of state x changes from a to b
- value_in(x)	The value of state x when entering the table

Example 3 (from Mortality.mpp)

In this example, a filter is set for a specific age, 80. The filter is true as long as a person is at age 80, thus

records a period of time and not a single snapshot, as in the previous example. As a consequence, `value_in(x)` and `value_out(x)` of a state can differ, as `x` might change while a person meets the condition of being age 80.

```
table Person StatisticsAtAge80 //EN Statistics at age 80
[ integer_age == 80 ]
{
    {
        unit, //EN Number of persons entering age 80
        value_in(alive), //EN Number of persons entering age 80
        value_out(alive), //EN Number of persons surviving age
80
        transitions(alive, TRUE, FALSE) //EN Number of persons dying at age 80
    } //EN Demographic characteristics
};
```

Example 4 (from Mortality.mpp)

In this example, we add a table dimension, `integer_age`, and produce a table by age. All formulas are calculated for each age. In our case, we used this for producing a validation table of simulated mortality rates by age, which can be compared to the input parameter.

```
table Person DeathRates //EN Death rates and
probabilities
{
    integer_age *
    {
        transitions(alive, TRUE, FALSE), //EN Number of persons dying
at this age
        transitions(alive, TRUE, FALSE) / duration(), //EN Death rate decimals=4
        transitions(alive, TRUE, FALSE) / unit //EN Death probability
decimals=4
    } //EN Demographic
characteristics
};
```

7.1.3. How to reproduce this modeling step

This is a ready-made template and nothing has to be added at this step.

Useful Visual Studio settings:

Syntax Highlighting

To switch on syntax highlighting for Modgen files (.mpp code and .dat data files):

- Go to: TOOLS / Options... / Text Editor / File Extensions
- Choose Editor / Microsoft Visual C++
- Add the following extensions (type into box and click add): mpp, dat

Switching off the squiggles – spell checker

By default, the Visual Studio underlines misspelled words with squiggles. This can be annoying in a programming language. To switch squiggles off:

- Go to: TOOLS / Options... / Text Editor / C++ / Advanced
- In the IntelliSense list, put 'Disable Squiggles TRUE'

7.1.4. Model versions

Converting the model to a template for time-based models

As at any modeling step of this guide, the case-based model can be converted to a time-based model by replacing the DYNAMIS-POP-MRT.mpp module, changing some settings in ModelSettings.mpp, and adapting the model parameters in the data files.

- Replace the module DYNAMIS-POP-MRT.mpp file by the time-based version of the file (see Appendix)
- Adapt the model settings in ModelSettings.mpp
- Adapt the data file containing the parameters of ModelSettings.mpp

Note that model conversion is discussed in detail in *Step 11: Conversion to a Time-Based Model*.

The module ModelSettings.mpp in a time-based version at Step 0

The essential code of this module consists in the declaration of one single parameter for the population sample size, as the scenario settings for time-based models do not contain such a parameter.

```
parameters
{
    long    TotalPopSampleSize;           //EN Simulated sample size
};
```

7.2. Step 2: Creating a Population

7.2.1. Overview

This step adds a module StartPopSampling.mpp to the model, which is used to sample the three characteristics of people typically found in macro population projection models: age, sex, and region. With this step we move from a cohort model, where all were born at the same time, to a population model, creating a population with characteristics as observed today.

Sampling from a distributional table allows the user to select any sample size; large samples will reduce monte carlo variation and results convert to those obtained from a cohort-component macro model. This approach works as long as two central limitations of cell-based models is satisfied: the number of characteristics is small and all characteristics are categorical, as the population has to be split into cells. In micro-simulation, this can be overcome by replacing a distributional table by a population micro-data file, an approach to which we will convert at a later point when extending the model.

While converting the model to a population model is straightforward, there are some issues to consider:

- As the distributional table reflects the population as observed today, when creating people in the past (starting their simulation at age 0), we have to disable mortality in the past, as today we only observe survivors. The alternative option would be to create the population as of today, thus people being born with their current age. We opted for the first option to create the starting population and will demonstrate the second option later for immigrants.
- Existing tables have to be carefully revised. For example, life expectancy, as currently calculated, has little meaning, as it now reflects the life expectancy calculated from people who survived until today.
- While most of the code can be added within the additional module StartPopSampling.mpp, we must make modifications to other modules, too. Besides Mortality.mpp, we also modify PersonCore.mpp:

At `Start()` we call the sampling function to obtain initial characteristics. Also, `StartPopSampling.mpp` is a good place to put some of the new type definitions and states, such as sex and region, as they are expected to be used in many other modules.

- We have two design options for ending a simulation: let everybody live until death, or set an end time when the simulation will be terminated and all actors killed. At this step we opt for the first option, setting the time horizon to 2113; as the youngest people in the starting population are born in 2012, we allow them to reach the maximum age of 100.9999 years.

7.2.2. Concepts

Overview

At this step, we introduce two key Modgen concepts and functionality:

- Adding a new module
- Sampling from a distributional table

Adding a new module

Adding a new module:

- Right-click `Modules(mpp)` in the solution explorer
- Select `add / new item / Modgen`
- Select `Modgen12EmptyModuleVide`
- Give the module a name

A new empty module of the chosen name appears in the list of mpp files in the solution explorer.

Sampling from a distributional parameter table

To allow sampling from a table, Modgen uses the parameter type `cumrate [n]`. A `cumrate` parameter is of value type `double` and allows sampling using a function `Lookup_TableName()`. For each `cumrate` parameter, such a function is available automatically. The `[n]` indicates the number of dimensions sampled; when smaller than the number of total dimensions, the last `n` dimensions are sampled. `Cumrate` parameters are declared like other parameter tables.

Example (From `StartPopSampling.mpp`)

```
parameters
{
    cumrate[3] StartingPopulationTable[SEX][AGE_RANGE][PROVINCE_NAT]; //EN Starting
    population
};
```

For sampling from a `cumrate` parameter table, the `Lookup_TableName` function is used. This is a C++ type function following some C++ syntax. The function takes a random number as a parameter, and integer variables for the table dimensions. The last `[n] - n` is set when declaring the parameter—characteristics are sampled, the local integer variables taking up the sampled values are introduced with the value sign `&`. In our case, all three dimensions are sampled.

As the `Lookup` function returns integer values for the table position, the sampled characteristics typically have to be cast in the types of the corresponding states. For example `'(SEX)nNumber'` converts the integer `nNumber` into a category of the classification `SEX`. (Levels of classifications correspond to integers starting at 0.)

```

void Person::GetStartCharacteristics()
{
    int nAge, nSex, nProv;

    Lookup_StartingPopulationTable(RandUniform(2), &nSex, &nAge, &nProv);

    integer_age = nAge // Age in full years
    sex = (SEX)nSex; // Sex
    province_nat = (PROVINCE_NAT)nProv; // Province of residence
}

```

If we sample age and province for given sex, the code would change as follows:

```

cumrate[2] DistributionTable[SEX][AGE_RANGE][PROVINCE_NAT]; //EN Distribution table
Lookup_DistributionTable(RandUniform(2), (int)sex, &nAge, &nProv);

```

7.2.3. How to reproduce this modeling step

Adding new types in PersonCore.mpp

Besides age, which was the only characteristic people had before, actors will get three more characteristics, which we will add in the PersonCore.mpp module: sex, year of birth, and region of residence. For that we define two classifications (one for sex, one for province) and a range for projected years (allowing the simulation to run from 2013 to 2113) and all years (1912-2113).

```

classification SEX //EN Sex
{
    FEMALE, //EN Female
    MALE //EN Male
};

classification PROVINCE_NAT //EN Province
{
    PN_PROV00, //EN Hodh-Charghy
    PN_PROV01, //EN Hodh-Gharby
    PN_PROV02, //EN Assaba
    PN_PROV03, //EN Gorgol
    PN_PROV04, //EN Brakna
    PN_PROV05, //EN Trarza
    PN_PROV06, //EN Adrar
    PN_PROV07, //EN Dakhlett-Nouadibou
    PN_PROV08, //EN Tagant
    PN_PROV09, //EN Guidimagha
    PN_PROV10, //EN Tirs-Ezamour
    PN_PROV11, //EN Inchiri
    PN_PROV12 //EN Nouakchott
};

range SIM_YEAR_RANGE{ 2013, 2113 }; //EN Projected calendar years
range ALL_YEAR_RANGE{ 1912, 2113 }; //EN All calendar years

```

New states in PersonCore.mpp

We add the following states:

- Year_of_birth: a derived state calculated automatically from the existing state time_of_birth by the function int(), which typecasts a double number to integer.

- Sex: a simple state initialized to { FEMALE } - the actual sex will be sampled at creation of each actor.
- Province_nat: region within the national territory (as opposed to regions including foreign). Another simple state initialized with the first province of the list: the actual province of residence will be sampled at creation of each actor.
- Calendar_year: a self-scheduling state based on time, within the range of past and projected years.
- Sim_year: a derived state based on calendar_year but forced into the range of projected years.
- In_projected_time: an indicator, in cases where the simulation has moved from the past to the projected time. This state can be used in tables, for example, to filter output to the future, allowing the state sim_year to be used as a table dimension.

```
actor Person
{
  ALL_YEAR_RANGE  year_of_birth = int(time_of_birth);           //EN Year of birth
  SEX              sex = { FEMALE };                           //EN Sex
  PROVINCE_NAT    province_nat = { PN_PROV00 };                //EN Province of
residence

  //EN Calendar year
  integer         calendar_year = self_scheduling_int(time);

  //EN Projected year
  SIM_YEAR_RANGE  sim_year = COERCE(SIM_YEAR_RANGE, calendar_year);

  //EN The simulation is within the projected time
  logical         in_projected_time = (calendar_year >= MIN(SIM_YEAR_RANGE)) ? TRUE
: FALSE;
};
```

Create a new module StartPopSampling.mpp

We add a new module StartPopSampling.mpp by:

- Right-click Modules(mpp) in the solution explorer
- Select add / new item / Modgen
- Select Modgen12EmptyModuleVide
- Name it StartPopSampling

A new empty module StartPopSampling.mpp appears in the list of mpp files in the solution explorer.

Code of the new module StartPopSampling.mpp

Documentation

As with any other module, the module starts with a label and note for documentation:

```
//LABEL(StartPopSampling, EN) Starting population sampling module

/* NOTE(StartPopSampling, EN)

  Added in Step 2

  This module implements the sampling of characteristics of the starting
population.
  At the creation of each person of the starting population, the characteristics
age, sex, and region are drawn from a distributional table
```

```
*/
```

Parameters

The module has one single parameter: the distributional table from which the characteristics are sampled at birth of an actor. To allow sampling from a table, Modgen uses the parameter type **cumrate**. A cumrate parameter is of value type double and allows sampling using a function `Lookup_TableName()`. For each cumrate parameter, such a function is available automatically. The `[n]` indicates the number of dimensions sampled; when smaller than the number of total dimensions, the last `n` dimensions are sampled.

```
parameters
{
    cumrate[3] StartingPopulationTable[SEX][AGE_RANGE][PROVINCE_NAT]; //EN Starting
    population
};

parameter_group PG01_StartPop //EN
Population
{
    StartingPopulationTable
};
```

Actor function

The actor `Person` block of this module contains a single function for sampling characteristics. The parameter `PersonNr` is not used at this step, but allows easy conversion of the model to a version that reads in a starting population file.

```
actor Person
{
    void GetStartCharacteristics(long PersonNr); //EN Sample characteristics,
    called at Start()
};
```

Function implementation

The `Lookup_TableName` function is used for sampling from the cumrate table. It takes a random number as a parameter and integer values for the table dimensions. The last `[n] - n` is set when declaring the parameter, characteristics are sampled, and the local integer variables taking up the sampled values are introduced with the value sign `&`. In our case, all three dimensions are sampled.

As the `Lookup` function returns integer values for the table position, the sampled characteristics have to be casted into the types of the corresponding states. For example `(SEX)nNumber` converts the integer `nNumber` into a category of the classification `SEX`. (The levels of classification correspond to integers starting at 0.)

```
void Person::GetStartCharacteristics(long PersonNr)
{
    int nAge, nSex, nProv;

    // Samling of characteristics sex, age and province from the cumrate table
    StartingPopulationTable
    // Cumrate parameters automatically provide a sampling function
    Lookup_ParameterName
    // The lookup function returns integer values

    Lookup_StartingPopulationTable(RandUniform(2), &nSex, &nAge, &nProv);

    // Storing the sampled variables into the corresponding states
    // Integer values can be casted to classifications by
```

```
(CLASSIFICATION_NAME)nIntegerVariable

sex = (SEX)nSex; // Sex
province_nat = (PROVINCE_NAT)nProv; // Province of residence

// Time of birth at a random moment within the year
time_at_birth = MIN(SIM_YEAR_RANGE) - nAge - RandUniform(3);
}
```

Modify the Start() function in PersonCore.mpp

The Start() function now calls the GetStartCharacteristics() function and sets the state time to the time_of_birth calculated in the sampling function from the sampled current age.

```
void Person::Start(long PersonNr, Person *prPerson)
{
    GetStartCharacteristics();
    age = 0;
    time = time_at_birth;
}
```

Modify timeMortalityEvent() in Mortality.mpp

While persons in the starting population are created in the past, their composition reflects today's population, so they are survivors until the start of the simulation and mortality has to be disabled in the past. This is accomplished by adding an additional condition: the logical state in_projected_time, when assessing if a person is at risk of death.

```
// check if a person is at risk
if ( MortalityTable[integer_age] > 0.0 && in_projected_time )
```

Adding the parameter and values in the Base(PersonCore).dat file

The parameter has $2 \times 101 \times 13 = 2626$ values (sex x age x province). Tables by age and province by sex can be easily produced from a micro-data file or are available from macro population projections, which take up the same parameter. Tables by sex are easy to copy and paste into the table in the graphical user interface. To assign initial values to the parameter, a (repeater) can be used:

```
cumrate[3] StartingPopulationTable[SEX][AGE_RANGE][PROVINCE_NAT] = { (2626) 1 };
```

Add and modify tables in Tables.mpp

Many cohort tables have limited meaning in the population model. We switched off mortality in the past, as people of the starting population are survivors.

At this step, we keep the validation table only for death rates, but add a filter [in_projected_time] to calculate rates in the projected time.

```
table Person DeathRates //EN Death rates and
probabilities
[ in_projected_time ]
{
    integer_age *
    {
        transitions(alive, TRUE, FALSE), //EN Number of persons
dying at this age
        transitions(alive, TRUE, FALSE) / duration(), //EN Death Rate
decimals=4
        transitions(alive, TRUE, FALSE) / unit //EN Death probability
    }
}
```

```

decimals=4
}
};

```

We add a second validation table for displaying the simulated starting population by age and province by sex. This table just counts people at the moment the simulation starts by placing the filter [trigger_entrances(in_projected_time, TRUE)].

```

table Person SimulatedStartingPopulation //EN Simulated starting
population
[ trigger_entrances(in_projected_time, TRUE) ]
{
  sex+ *
  {
    unit //EN Number of persons
  }
  * integer_age+
  * province_nat+
};

```

As we are not able to calculate the life expectancy at birth because the starting population contains only survivors up to date, we remove all other previous tables and replace them by a table for the remaining life expectancy by age at the start of the simulation. This age can be programmed as a derived state, as it is used only for the table, and is placed in the Tables.mpp file.

```

actor Person
{
  //EN Age at start
  AGE_RANGE tab_age_start = value_at_transitions(in_projected_time, FALSE, TRUE,
integer_age);
};

table Person RemainingLifeExpectancyAge //EN Remaining life expectancy by
age at start
[ in_projected_time ]
{
  tab_age_start *
  {
    duration() / unit //EN Remaining life expectancy in
simulation decimals=4
  }
};

```

7.2.4. Model versions

Converting the model to a time-based and/or micro-data-based version

As in any modeling step in this guide, the case-based model can be converted to time-based by replacing the DYNAMIS-POP-MRT.mpp module (see Appendix for code) and adapting its settings in ModelSettings.mpp.

As in any step starting from 1, the starting population type can be converted from a sampling table to a micro-data file by replacing the starting population file (see Appendix for code) and adapting the settings in ModelSettings.mpp.

Note that model conversion is discussed in detail in *Step 11: Conversion to a Time-Based Model*.

7.2.5. The module ModelSettings.mpp in alternative versions of Step 1

Sampling - time based

```
parameters
{
    long    TotalPopSampleSize;           //EN Simulated sample size
};
```

Micro-data file - case based

```
parameters
{
    model_generated long    StartPopSampleSize;           //EN Sample size of starting
population
    double                  StartPopSize;                //EN Total population size

    model_generated long    TotalPopSampleSize;           //EN Sample size of starting
population
    model_generated double  TotalPopSize;                //EN Total population size

    model_generated logical ScalePopulation;              //EN Scale population
    model_generated double  ActorWeight;                 //EN Actor Weight
};

void PreSimulation()
{
    //Total population
    TotalPopSize = StartPopSize;
    TotalPopSampleSize = GetAllCases();
    StartPopSampleSize = TotalPopSampleSize;

    // Weight
    ActorWeight = TotalPopSize / TotalPopSampleSize;

    //Scale Population
    ScalePopulation = GetPopulationScalingRequired();
    if (ScalePopulation) SetPopulation(TotalPopSize);
};
```

Micro-data file - time based

```
parameters
{
    model_generated long    StartPopSampleSize;           //EN Sample size of starting
population
    double                  StartPopSize;                //EN Total population size
    long                    TotalPopSampleSize;           //EN Sample size of starting
population
    model_generated double  TotalPopSize;                //EN Total population size
};

parameter_group PG_ModelSettings                       //EN Model Settings
{
    StartPopSize, TotalPopSampleSize, ScalePopulation
};

void PreSimulation()
```

```
{
    TotalPopSize = StartPopSize;
    StartPopSampleSize = TotalPopSampleSize;
};
```

7.3. Step 3: Population Scaling

7.3.1. Overview

This step adds automatic population scaling. The user can select both the sample size and real population size and the model output is automatically scaled to the real population size. While this feature is automatically provided by Modgen in case-based models (the population sizes and a scaling option are part of the model settings), in time-based models, parameters for scaling and scaling has to be handled in the code. In the case-based version of DYNAMIS-POP-MRT, information on total population size is already contained in the distributional table; we will introduce a routine that uses this information and overwrites the parameter settings with the calculated value.

7.3.2. Concepts

How to scale model output

Modgen provides a series of useful functions for handling population scaling (weighting). In DYNAMIS-POP-MRT, all persons have the same weight and scaling can be switched on and off.

Scaling in case-based models

The user can set three parameters in the scenario settings: the number of cases, target population size, and a box to check if population scaling is requested. This information can be accessed by the following functions:

```
GetPopulationScalingRequired(); // returns true if the checkbox is clicked
GetAllCases(); // returns the number of cases to be
simulated
GetPopulation(); // returns the total target population
size
```

If population scaling is clicked on, scaling is switched on automatically and no more programming is required. In some cases, like in DYNAMIS-POP-MRT, it is useful to overwrite the total population size, as the information may be contained in parameters. In this case, population size can be set in the code and no more programming is required: if population scaling is switched on, the target size is automatically used:

```
SetPopulation(SizeOfThePopulation); // Sets the total target population size
```

A good place to put this function is in PreSimulation().

Scaling in time-based models

Time-based models do not contain scenario settings for population scaling, and the SetPopulation() function has no effect in time-based models. Additional parameters therefore must be added to the model. Once the weight of an actor is calculated, it has to be set by the following functions:

```
Set_actor_weight (ActorWeight);
Set_actor_subsample_weight (ActorWeight);
```

A good place to set the actor weight is in the Start() function.

7.3.3. How to reproduce this step

- We add a model-generates parameter in StartPopSampling.mpp.
- We add code to ModelSettings.mpp for making scenario settings transparent and allow population scaling both in case-based and time-based model versions.
- We modify the Start() function to weight actors in time-based models.

Changes in StartPopSampling.mpp

We change the starting population sampling table to a model_generated parameter and add a population table of the same shape as a normal parameter. This allows us to calculate the size of the starting population from the parameter table in the pre-simulation, and this enables scaling the simulation and determining the probability that an actor comes from the starting population versus the immigrant population. This cannot be done directly from a cumrate table, as Modgen optimizes and re-scales cumrate tables, with the information of the original cell values lost. To solve this problem, we count values and copy them over from the parameter table to the model-generated cumrate table.

```
parameters
{
    double          StartingPopulation[SEX][AGE_RANGE][PROVINCE_NAT];          //EN
Starting population
    //EN Starting population
    model_generated cumrate[3]
StartingPopulationTable[SEX][AGE_RANGE][PROVINCE_NAT];  };

parameter_group PG01_StartPop //EN Starting Population Characteristics
{
    StartingPopulation
};
```

Code in the module ModelSettings.mpp

We add a series of model-generated parameters that can be set in a PreSimulation() function of the model and are not visible to the user. In the case-based model version, some of these parameters will be set by scenario settings, while time-based models will have to make these parameters real parameters, set by the user. Making all parameters explicit and placing them into one file makes the model more transparent and simplifies model conversions.

Parameters:

```
parameters
{
    model_generated long      StartPopSampleSize;          //EN Sample size of starting
population
    model_generated double    StartPopSize;                //EN Total population size

    model_generated long      TotalPopSampleSize;          //EN Sample size of starting
population
    model_generated double    TotalPopSize;                //EN Total population size

    model_generated logical   ScalePopulation;             //EN Scale population
    model_generated double    ActorWeight;                 //EN Actor Weight
};
```

Pre-Simulation

In the case-based version of DYNAMIS-POP-MRT, all information is contained in the scenario settings. As the total population size is already known from the sampling table parameter, which has population totals by age, sex, and province, we can add up the totals in the table and overwrite the parameter in the scenario settings as it became redundant. As we use the population table for sampling, we need a table of type `cumrate`. Unfortunately, `cumrate` parameters do not allow access to their values, as the tables are internally optimized for sampling. One could use the inelegant work-around of copying values on a table of type `double` into a model-generated `cumrate` table of the same shape.

At the end of the function, when requested by the user and when the model is case-based (the parameter `model_is_case_based` is provided by the simulation engine), we set the target population, which automatically scales the population, in case-based models. We also calculate the actor weight “manually,” as this information will be used by time-based model versions in the `Start()` function.

```
void PreSimulation()
{
    // Starting population
    StartPopSize = 0.0;
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
        {
            for (int nProv = 0; nProv < SIZE(PROVINCE_NAT); nProv++)
            {
                StartPopSize = StartPopSize + StartingPopulation[nSex][nAge][nProv];
                StartingPopulationTable[nSex][nAge][nProv] =
                StartingPopulation[nSex][nAge][nProv];
            }
        }
    }

    //Total population
    TotalPopSize = StartPopSize;
    TotalPopSampleSize = GetAllCases();
    StartPopSampleSize = TotalPopSampleSize;

    // Weight
    ActorWeight = TotalPopSize / TotalPopSampleSize;

    //Scale Population
    ScalePopulation = GetPopulationScalingRequired();
    if (ScalePopulation && model_is_case_based)
    {
        SetPopulation(TotalPopSize);
    }
};
```

Changes in `PersonCore.mpp`

We add the following code for population scaling in the `Start()` function:

```
// Population scaling
if (ScalePopulation && !model_is_case_based)
{
    Set_actor_weight(ActorWeight);
    Set_actor_subsample_weight(ActorWeight);
}
```

This code is executed in time-based models only.

7.3.4. Model versions

Converting the model to a time-based and/or micro-data based version

As in any modeling step in this guide, the case-based model can be converted to time-based by replacing the DYNAMIS-POP-MRT.mpp module (see Appendix for code) and adapting its settings in ModelSettings.mpp.

As in any step starting from Step 1, the starting population type can be converted from a sampling table to a micro-data file by replacing the starting population file (see Appendix for code) and adapting the settings in ModelSettings.mpp.

Note that model conversion is discussed in detail in *Step 11: Conversion to a Time-Based Model*.

The module ModelSettings.mpp in alternative versions of Step 2

Sampling - time based

```

parameters
{
    model_generated long    StartPopSampleSize;           //EN Sample size of starting
population
    model_generated double  StartPopSize;                 //EN Total population size

    long                    TotalPopSampleSize;          //EN Sample size of starting
population
    model_generated double  TotalPopSize;                 //EN Total population size

    logical                  ScalePopulation;             //EN Scale population
    model_generated double  ActorWeight;                 //EN Actor Weight
};

parameter_group PG00_ModelSettings                       //EN Model Settings
{
    TotalPopSampleSize, ScalePopulation
};

void PreSimulation()
{
    // Starting population
    StartPopSize = 0.0;
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
        {
            for (int nProv = 0; nProv < SIZE(PROVINCE_NAT); nProv++)
            {
                StartPopSize = StartPopSize + StartingPopulation[nSex][nAge][nProv];
                StartingPopulationTable[nSex][nAge][nProv] =
StartingPopulation[nSex][nAge][nProv];
            }
        }
    }

    //Total population
    TotalPopSize = StartPopSize;
    StartPopSampleSize = TotalPopSampleSize;

    // Weight
    ActorWeight = TotalPopSize / TotalPopSampleSize;

```

```
};
```

Micro data file - case based

```
parameters
{
    model_generated long    StartPopSampleSize;    //EN Sample size of starting
population
    double                StartPopSize;           //EN Total population size

    model_generated long    TotalPopSampleSize;    //EN Sample size of starting
population
    model_generated double  TotalPopSize;          //EN Total population size

    model_generated logical ScalePopulation;       //EN Scale population
    model_generated double  ActorWeight;           //EN Actor Weight
};

void PreSimulation()
{
    //Total population
    TotalPopSize = StartPopSize;
    TotalPopSampleSize = GetAllCases();
    StartPopSampleSize = TotalPopSampleSize;

    // Weight
    ActorWeight = TotalPopSize / TotalPopSampleSize;

    //Scale Population
    ScalePopulation = GetPopulationScalingRequired();
    if (ScalePopulation && model_is_case_based)
    {
        SetPopulation(TotalPopSize);
    }
};
```

Micro data file - time based

```
parameters
{
    model_generated long    StartPopSampleSize;    //EN Sample size of starting
population
    double                StartPopSize;           //EN Total population size

    long                  TotalPopSampleSize;    //EN Sample size of starting
population
    model_generated double  TotalPopSize;          //EN Total population size

    logical                ScalePopulation;       //EN Scale population
    model_generated double  ActorWeight;           //EN Actor Weight
};

parameter_group PG_ModelSettings //EN Model Settings
{
    StartPopSize, TotalPopSampleSize, ScalePopulation
};

void PreSimulation()
{
    //Total population
```

```

    TotalPopSize = StartPopSize;
    StartPopSampleSize = TotalPopSampleSize;

    // Weight
    ActorWeight = TotalPopSize / TotalPopSampleSize;
};

```

7.4. Step 4: Fertility

7.4.1. Overview

This step adds the module `Fertility.mpp` to the model. The module has three parameters: the distribution of age-specific fertility, total fertility rate, and sex ratio. The first two parameters are used to internally calculate age-specific fertility rates in the pre-simulation function, which is called before the start of the simulation. It is a so-called model-generated parameter not visible to users.

At the birth event, a new actor is created and the `Start()` function of this actor is called. To pass on a link to the mother, which is used by the child to determine its province, time of birth, etc., the `Start()` function is modified, now taking on a parameter—a link to the mother.

7.4.2. Concepts

Overview

At this step, we introduce two key Modgen concepts and functions:

- RANGE_POS
- Model-generated parameters
- Pre-simulation
- Creating an actor and inheriting characteristics

RANGE_POS

`RANGE_POS(NAME_RANGE, value)` returns the position of the value within a range, the first position being 0. This is used to address values within a range.

Example:

```

Range:      range FERTILE_AGE_RANGE { 15,49 };
Parameter:  double FertilityRate[FERTILE_AGE_RANGE];
derived state: double current_fertility = ( WITHIN(FERTILE_AGE_RANGE, integer_age) ) ?
          FertilityRate[RANGE_POS(FERTILE_AGE_RANGE, integer_age)] : 0.0;

```

As the fertile age range starts at 15, to get the right index for a given age in full years, we use the `RANGE_POS` function, which returns the index of a specific value within a range. For example, `RANGE_POS(FERTILE_AGE_RANGE, 15)` returns 0, as 15 is the first value in the range.

Model-generated parameters

Model-generated parameters are not visible to users, and are calculated in the Pre-Simulation function from the other parameters. Model-generated parameters are declared like other parameters, but headed by `model_generated`.

```

//EN Age specific fertility rate
model_generated double AgeSpecificFertilityRate[FERTILE_AGE_RANGE][SIM_YEAR_RANGE];

```

Pre-Simulation

The function **PreSimulation()** is called before the start of the simulation and before actors are created, so that new “model_generated” parameters can be calculated and other initialization can be made. PreSimulation() functions can be placed in the code multiple times, allowing to code module-specific pre-simulations within the module to which it belongs.

Creating and actor and inheriting characteristics

Creating and initializing a new actor requires only two lines of C++ code, which may look esoteric to non-C++ programmers. The first creates a new actor by declaring a pointer to an object of type Person created by the keyword new. The pointer can then be used to call the Start() function of the new Person, addressing it with “->”. Start functions allow parameters, in our case, of type “pointer to a person.” When calling the function with “this” as a parameter, a pointer to the calling person (i.e., the mother) is passed to the child. Within the Start function, the child can now make use of the pointer to access characteristics of the mother.

```
actor Person
{
    void    Start(long PersonNr, Person *peMother); // Start function with a
parameter 'pointer to a person'
};

void Person::BirthEvent()
{
    Person *peChild = new Person;           // Create and point to a new actor
    peChild->Start(-1, this);               // Call Start() of the new actor and pass own
address
}

void Person::Start(long PersonNr, Person *peMother)
{
    [...]
    time_of_birth = peMother->time;        // The time of creation
    [...]
}
```

7.4.3. How to reproduce this modeling step

Most of the code of this step is contained in a new module, Fertility.mpp.

The module Fertility.mpp

Add a new module Fertility.mpp

To add Fertility.mpp, the module will follow a typical organization of code: - Documentation - Dimensions - Parameters - Actor declarations - Function / Event implementations - pre-simulation routines.

Documentation

It is good practice to start with the label and a note describing the module.

```
//LABEL(Fertility, EN) Fertility

/* NOTE(Fertility, EN)

    Added at step 3

    This module implements fertility.
    Fertility is based on age-specific fertility rates calculated from two
```

```

parameters:
    an age distribution of fertility and the Total fertility rate TFR for future
years
    Another parameter is the sex-ratio
*/

```

New type definitions

The module introduces two additional ranges: one for the number of children, which we limit to 15, and the second for the fertile age range 15-49.

```

range FERTILE_AGE_RANGE{ 15, 49 };           //EN Fertile age range
range PARITY_RANGE{ 0, 15 };                 //EN Parity range

```

Parameters

The model has three parameter tables plus an internal model-generated parameter calculated from the other parameters. Model-generated parameters are calculated in the Pre-Simulation phase as described below. They are not visible to users. Model-generated parameters are declared like other parameters, but headed by `model_generated`.

```

parameters
{
    double AgeSpecificFertility[FERTILE_AGE_RANGE][SIM_YEAR_RANGE]; //EN Age
distribution of fertility
    double TotalFertilityRate[SIM_YEAR_RANGE]; //EN Total
fertility rate
    double SexRatio[SIM_YEAR_RANGE]; //EN Sex ratio

    //EN Age specific fertility rate
    model_generated double
AgeSpecificFertilityRate[FERTILE_AGE_RANGE][SIM_YEAR_RANGE];
};

parameter_group PG03_Fertility //EN Fertility
{
    AgeSpecificFertility, TotalFertilityRate, SexRatio
};

```

Actor declarations

The module introduces two simple states: parity and an indicator that a person has “started life,” i.e., left the `Start()` function. Parity simply counts the number of children and is incremented in the birth event.

The `SetAlive` Event is used to switch the state `set_alive` to true immediately after an actor is created and has processed its `Start()` function. Transitions are not visible if a state is changed/initialized in `Start()`. The state is used, for example, in tables for counting births, transitions(`set_alive, FALSE, TRUE`), and (later) to avoid linkages between actors before their time of birth is known, which is immediately after `Start()`.

```

actor Person
{
    PARITY_RANGE    parity = { 0 };           //EN Number of children born
    logical set_alive = { FALSE };           //EN Person is set to be alive already

    event    timeBirthEvent, BirthEvent;     //EN Birth event
    event    timeSetAliveEvent, SetAliveEvent; //EN Event to set set_alive to true
};

```

Event implementation

Birth is a typical event based on hazard rates. In the time function, it is checked if a person is at risk, and if

so, a waiting time is drawn from an exponential distribution based on the age and period-specific hazard. When a birth happens, the state parity is incremented by 1 (in C++ this can be done by `parity++`). The last two lines create a new child actor; this is C++ notation for creating a pointer to a new object and invoking a function—here, the `Start()` function—of the new actor. To establish communication between mother and child, the mother passes on her own address (“self”) to the child. Note that in C++, `Object->Function()` indicates that the function of another object (here, the child) is called.

```

TIME Person::timeBirthEvent()
{
    double dEventTime = TIME_INFINITE;
    if (sex==FEMALE && WITHIN(FERTILE_AGE_RANGE, integer_age) && parity <
MAX(PARITY_RANGE)
        && WITHIN(SIM_YEAR_RANGE, calendar_year))
    {
        double dHazard = AgeSpecificFertilityRate[RANGE_POS(FERTILE_AGE_RANGE,
integer_age)]
[RANGE_POS(SIM_YEAR_RANGE, calendar_year)];
        if (dHazard > 0.0) dEventTime = WAIT(-TIME(log(RandUniform(4)) / dHazard));
    }
    return dEventTime;
}

void Person::BirthEvent()
{
    parity++; // increment parity
    Person *peChild = new Person; // Create and point to a new actor
    peChild->Start(-1,this); // Call Start() of the new actor and pass own
address
}

```

Note that when addressing `AgeSpecificFertilityRate`, we do not use `integer_age` and `calendar_year` directly as index, but write:

```

AgeSpecificFertilityRate[RANGE_POS(FERTILE_AGE_RANGE,
integer_age)][RANGE_POS(SIM_YEAR_RANGE, calendar_year)]

```

This is valid, as ranges in Modgen internally start with index 0. As the fertile age range starts at 15 (range `FERTILE_AGE_RANGE { 15,49 }`); to get the right index for a given age in full years, we use the `RANGE_POS` function, which returns the index of a specific value within a range; for example, `RANGE_POS(FERTILE_AGE_RANGE, 15)` returns 0, as 15 is the first value in the range.

The `SetAliveEvent` is straightforward: if a person is not set alive yet, this is done immediately. In this way, the event occurs immediately after the `Start()` function was processed and the change in state can be observed, which will be useful for creating tables counting the number of births.

```

TIME Person::timeSetAliveEvent() { if (!set_alive) return WAIT(0); else return
TIME_INFINITE; }
void Person::SetAliveEvent() { set_alive = TRUE; }

```

Pre-Simulation

The function `PreSimulation()` is called before the start of the simulation and before actors are created. This way, new “model_generated” parameters can be calculated, and other initialization can be made. `PreSimulation()` functions can be placed multiple times in the code, allowing to code module-specific pre-simulations within the module to which it belongs.

In our case, we perform the calculation of age-specific fertility rates for each projected year in this function. The parameter is built by scaling the known age distribution of the period to total 1, and then multiplying it by the known total fertility rate of the period.

```

void PreSimulation()
{
    double dSum;
    for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
    {
        dSum = 0.0;
        // check if distribution parameter adds up too 1
        for (int nAge = 0; nAge < SIZE(FERTILE_AGE_RANGE); nAge++)
        {
            dSum = dSum + AgeSpecificFertility[nAge][nYear];
        }
        // scale distribution to 1 and convert to fertility rates; copy to model
        generated parameter
        for (int nAge = 0; nAge < SIZE(FERTILE_AGE_RANGE); nAge++)
        {
            if (dSum > 0.0)
            {
                AgeSpecificFertilityRate[nAge][nYear]
                    = AgeSpecificFertility[nAge][nYear] / dSum *
TotalFertilityRate[nYear];
            }
            else AgeSpecificFertilityRate[nAge][nYear] = 0.0;
        }
    }
}

```

Modification of the Start() Function in PersonCore.mpp

Using the parameter of the Start() function

As we pass on the address of the mother to the child at birth, we make use of one of the two parameters of the start function:

```

void Start(long PersonNr, Person *peMother); //EN Starts the actor

```

The parameter is also used to distinguish whether a person is created from the starting population (in this case the mother is unknown and NULL will be used as parameter) or created at a birth event and thus knows her mother. In the second case, the characteristics at birth are not sampled as before—by calling GetStartCharacteristics()—but initialized within the Start function.

- Sex is randomly chosen based on the given sex ratio
- Time_of_birth is set by accessing the mother's state time
- Province_nat is inherited from the mother

```

void Person::Start(Person *peMother)
{
    age = 0;
    if (peMother == NULL) // No mother: the person comes from the starting
    population
    {
        GetStartCharacteristics();
    }
    else // There is a mother: the person is a child born in
    the simulation
    {
        // assign a sex according to the sex ratio parameter
        sex = MALE;
    }
}

```

```

    if (RandUniform(5) < 100.0 / (100.0 + SexRatio[RANGE_POS(SIM_YEAR_RANGE,
calendar_year]))
    {
        sex = FEMALE;
    }
    // inherit characteristics from the mother
    time_of_birth = peMother->time; // The time of creation
    province_nat = peMother->province_nat; // The province at birth
}
time = time_at_birth;
}

```

Add parameters to the data file and initialize them with values

The model code is now complete. To run the model, we still have to add the parameters and parameter values to the data file. If values will be copied and pasted later, dummy tables can be generated by:

```

double AgeSpecificFertility[FERTILE_AGE_RANGE][SIM_YEAR_RANGE] = {(100)1};
double TotalFertilityRate[SIM_YEAR_RANGE] = {(100) 4};
double SexRatio[SIM_YEAR_RANGE] = {(100) 106.0};

```

Tables

We add two tables to the output, one for age-specific fertility rates, the second for number of births.

```

// A New table group for fertility tables

table_group TG_03_FertilityTables //EN Fertility Tables
{
    FertilityRates, NumberBirths
};

// A state used for the following table

actor Person
{
    FERTILE_AGE_RANGE tab_fert_age = COERCE(FERTILE_AGE_RANGE, integer_age); //EN Age
};

// Tables

table Person FertilityRates //EN Fertility rates
[WITHIN(SIM_YEAR_RANGE, calendar_year) && WITHIN(FERTILE_AGE_RANGE, integer_age) &&
sex == FEMALE]
{
    {
        parity / duration() //EN Fertility rates
    }
    decimals=4
    * tab_fert_age
    * sim_year
};

table Person NumberBirths //EN Number of births
[WITHIN(SIM_YEAR_RANGE, calendar_year)]
{
    {
        entrances(set_alive, TRUE) //EN Births
    }
};

```

```

    }
    * sim_year
    * sex +
};

```

Modifications of an existing table

We add a filter condition to include those born in the future from the life expectancy calculation.

```

table Person RemainingLifeExpectancyAge //EN Remaining life expectancy
by age at start
[in_projected_time && time_of_birth < MIN(SIM_YEAR_RANGE)]
{
    tab_age_start *
    {
        duration() / unit //EN Remaining life expectancy in
simulation decimals=4
    }
};

```

7.4.4. Model versions

Converting the model to a time-based and/or micro-data-based version

As in any modeling step of this guide, the case-based model can be converted to time-based by replacing the DYNAMIS-POP-MRT.mpp module (see Appendix for code) and adapting its settings in ModelSettings.mpp.

As in any step, starting from Step 1, the starting population type can be converted from a sampling table to a micro-data file by replacing the starting population file (see Appendix for code) and adapting the settings in ModelSettings.mpp.

No changes affecting model conversions are made at this step, compared to the previous step. Note that model conversion is discussed in detail in *Step 11: Conversion to a Time-Based Model*.

7.5. Step 5: Refining Mortality

7.5.1. Overview

This step refines the mortality module. The current life table is extended to model mortality by sex. The mortality table interprets a standard life table, which is used for modeling the age-specific mortality patterns, while we add a second parameter for life expectancy at birth. In the pre-simulation phase, a calibration factor for each simulation year is searched for, which, when applied as proportional factor, adjusts the standard mortality table to produce the target life expectancy.

7.5.2. Concepts

This step does not introduce new Modgen concepts, but does introduce some C++ programming within a PreSimulation function performing a binary search for finding calibration factors. This scales the mortality table to produce the target period life expectancies, which are a model parameter.

Example for a binary search

This binary search algorithm is used to find calibration factors that will be copied into a model-generated trend parameter.

```

void PreSimulation()
{
    double dLower, dUpper, dCenter, dResult, dTarget, dAlive, dDeaths;
    int nIterations;
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
        {
            dTarget = LifeExpectancy[nYear][nSex]; // Target: life expectancy
            dResult = 0.0; // Search result: life expectancy
            nIterations = 10000; // Maximum number of iterations
            dLower = 0.1; // Lower limit of calibration factor (relative risk)
            dUpper = 3.0; // Upper limit of calibration factor (relative risk)

            while (abs(dResult - dTarget) > 0.0001 && nIterations > 0)
            {
                nIterations--;
                dCenter = (dLower + dUpper) / 2.0; // New calibration factor for probing
                dResult = 0.0;
                dAlive = 1.0; // Proportion of people still alive

                // Life expectancy calculated applying calibration factor
                for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
                {
                    // proportion of deaths in year: survival = exp(-hazard)
                    dDeaths = dAlive * (1 - exp(-MortalityTable[nAge][nSex] *
                    dCenter));
                    dAlive = dAlive - dDeaths;
                    // people dying in this year are assumed to die in the middle of
                    the year
                    dResult = dResult + dDeaths * 0.5 + dAlive;
                }
                // Moving the search limits for next iteration
                if (dTarget < dResult) dLower = dCenter;
                else dUpper = dCenter;
            }
            // copying the best solution into the model-generated mortality trend
            MortalityTrend[nYear][nSex] = dCenter;
        }
    }
}

```

7.5.3. How to reproduce this modeling step

All code changes at this step are made in the existing module, `Mortality.mpp`, and the data file.

- Update the documentation
- Extend the standard mortality table by an additional dimension, sex
- Add a parameter for life expectancy

- Add a model-generated parameter for calibration factors reflecting mortality trends
- Change the time function of mortality to model mortality by sex and apply trend adjustments
- Calculate (find by binary search) the model-generated parameter MortalityTrend in PreSimulation()
- Add parameters with values to .dat file
- Modify tables

Code changes and new code in Mortality.mpp

Update the documentation

```

/* NOTE (Mortality, EN)

    Changed at step 4

    This module implements a simple model of mortality.
    People can die at any moment of time based on age-specific mortality rates.
    Age specific mortality rates are calculated from a standard life table and a
    trend factor.
    The trend factor is found in pre-simulation and calibrates mortality to reach a
    target
    period life expectancy. The standard life table and target life expectancies are
    model parameters.
    The only state of this module is 'alive' which is set to 'false' at death.
    Also at death, the Modgen function Finish() is called which clears up memory
    space.
*/

```

Update parameters

```

parameters
{
    double          MortalityHazard[AGE_RANGE][SEX];          //EN Mortality
    hazard by age
    double          LifeExpectancy[SIM_YEAR_RANGE][SEX];     //EN Life
    Expectancy
    model_generated double MortalityTrend[SIM_YEAR_RANGE][SEX]; //EN Mortality
    trend
};

parameter_group PG02_Mortality                               //EN Mortality
{
    MortalityTable, LifeExpectancy
};

```

As with any parameter changes, parameters also have to be updated in the data file.

Update the mortality time function

To apply the new mortality hazard, calculated by mortality, age, sex, and the trend factor, we add a new local variable, **dMortalityHazard**, and initialize it with the formula. We then use dMortalityHazard instead of the previous one-dimensional parameter in the code, where the hazard is needed to check if a person is at risk, and to calculate a waiting time.

```

TIME Person::timeMortalityEvent ()
{
    TIME    dEventTime = TIME_INFINITE;
    double  dMortalityHazard
            = MortalityTable[integer_age][sex] * MortalityTrend[RANGE_POS(SIM_YEAR_RANGE,

```

```

calendar_year)][sex];

    // check if a person is at risk
    if (dMortalityHazard > 0.0 && in_projected_time)
    {
        // determine the event time
        // the formula [ -log(rand) / hazard ] calculates an exponentially
distributed waiting time
        // based on a uniform distributed random number and a given hazard rate
        dEventTime = WAIT(-log(RandUniform(1)) / dMortalityHazard);
    }
    // return the event time, if the maximum age is not reached at that point
    if (dEventTime < time_of_birth + MAX(AGE_RANGE) + 1.0) return dEventTime;
    // otherwise, return the moment, at which the maximum age is reached
    else return time_of_birth + MAX(AGE_RANGE) + 0.9999;
}

```

Calculate the mortality trend adjustment factors in pre-simulation

This binary search algorithm is used to find calibration factors, which are copied into a model-generated trend parameter MortalityTrend[nYear][nSex].

```

void PreSimulation()
{
    double dLower, dUpper, dCenter, dResult, dTarget, dAlive, dDeaths;
    int nIterations;
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
        {
            dTarget = LifeExpectancy[nYear][nSex]; // Target: life expectancy
            dResult = 0.0; // Search result: life
expectancy
            nIterations = 10000; // Maximum number of
iterations in search
            dLower = 0.1; // Lower limit of calibration factor
(relative risk)
            dUpper = 3.0; // Upper limit of calibration factor
(relative risk)

            while (abs(dResult - dTarget) > 0.0001 && nIterations > 0)
            {
                nIterations--;
                dCenter = (dLower + dUpper) / 2.0; // New calibration factor for
probing

                dResult = 0.0;
                dAlive = 1.0; // Proportion of people still
alive

                // Life expectancy calculated applying calibration factor
                for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
                {
                    // proportion of deaths in year: survival = exp(-hazard)
                    dDeaths = dAlive * (1 - exp(-MortalityTable[nAge][nSex] *
dCenter));

                    dAlive = dAlive - dDeaths;
                    // people dying in this year are assumed to die in the middle of
the year

                    dResult = dResult + dDeaths * 0.5 + dAlive;

```

```

    }
    // Moving the search limits for next iteration
    if (dTarget < dResult) dLower = dCenter;
    else dUpper = dCenter;
  }
  // copying the best solution into the model-generated mortality trend
parameter
  MortalityTrend[nYear][nSex] = dCenter;
}
}
}
}

```

Changes in Tables.mpp

Because we added sex to mortality and mortality follows a time trend, we modify the table DeathRates to add the two new dimensions. In its default view (users can always select the ordering of dimensions in the user interface), the table now displays mortality measures by age and year; the other two dimensions of displayed measure (number, rates, probabilities) and sex (adding + also produces totals) can be selected from drop-down lists.

```

table Person DeathRates //EN Death rates and
probabilities
[ in_projected_time ]
{
  sex+ *
  {
    transitions(alive, TRUE, FALSE), //EN Number of persons
dying at this age
    transitions(alive, TRUE, FALSE) / duration(), //EN Death Rate decimals=4
    transitions(alive, TRUE, FALSE) / unit //EN Death probability
decimals=4
  }
  * integer_age
  * sim_year
};

```

7.5.4 Model versions

Converting the model to a time-based and/or micro-data-based version

As in any modeling step of this guide, the case-based model can be converted to time-based by replacing the DYNAMIS-POP-MRT.mpp module (see Appendix for code) and adapting its settings in ModelSettings.mpp.

As in any step starting from Step 1, the starting population type can be converted from a sampling table to a micro-data file by replacing the starting population file (see Appendix for code) and adapting the settings in ModelSettings.mpp.

No changes affecting model conversions are made in this step, compared to the previous step. Note that model conversion is discussed in detail in *Step 11: Conversion to a Time-Based Model*.

7.6. Step 6: Migration

7.6.1. Overview

This step adds internal migration to the model. It is based on age- and sex-specific transition matrices.

Only one transition per year is allowed. The module has three parameters, one to switch migration on/off, the second for the probability to move away (by province, age, and sex), and the third to sample the destination province (by origin, age, and sex).

7.6.2. Concepts

- Transition matrices
- SPLIT
- Converting probabilities to rates

Transition matrices

Transition matrices are origin-destination matrices containing probabilities to move from one to another state in a given time period. They are typically divided into two parameters, one containing the probability to leave by origin, and the second the distribution of destinations by origin. The second parameter—using the Modgen type `cumrate`—allows sampling the destination when the event happens.

Example

```
parameters
{
    double MigrationProbability[SEX] [AGE5_PART] [PROVINCE_NAT]; //EN
Migration probability
    cumrate MigrationDestination[SEX] [PROVINCE_NAT] [AGE5_PART] [PROVINCE_NAT]; //EN
Migration Destination
};
```

As such matrices contain probabilities to move within a period of time (usually years), there are four ways of implementing events:

- By a clock event creating an event at a specific point in each period, e.g., in the middle of the year. At that time, it is decided if an actor moves and, if so, the move to a sampled destination is done immediately. Following this approach, everybody moves at the same time if modeling moves by calendar year.
- Deciding at the beginning of each period if a person is supposed to move. If so, a random time within the period is drawn and the event is scheduled for this time. One drawback is that characteristics that may influence the probability to move may change within the period.
- Probabilities are converted to corresponding hazard rates [$\text{rate} = -\ln(1-\text{probability})$] and the event is implemented in continuous time. Using this approach, as here in the `Migration.mpp` module, one has to be sure that only one event can happen within the period.
- The transition matrix of probabilities is converted to a matrix of rates. Applying this approach, multiple events are possible within a year, but after each period, consistency with the original origin-destination transitions is maintained. There exists a Taylor series approach for this approach. The beauty of this approach is that transition matrices calculated from longer intervals, e.g., five-year inter-census intervals, can also be used without restricting moves to a single point in time per interval.

SPLIT(), split(), self_scheduling_split()

Split functions break up continuous variables into intervals.

`split()` can be used in tables or for derived states; it is dynamic, thus maintains itself:

Example

```

partition AGE5_PART { 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60 }; //EN Age
Groups

table Person TabNumberMoves //EN Internal migration rate
[WITHIN(SIM_YEAR_RANGE, calendar_year)]
{
    {
        number_moves / duration() //EN Migration rate decimals=4
    }
    * split(integer_age, AGE5_PART) //EN Age group
};

```

SPLIT() is used within functions. It is static, returning the index at a point in time.

Example

```

void Person::MigrationEvent()
{
    int nDestination;
    int nAge5 = SPLIT(integer_age, AGE5_PART);
    [...]
}

```

self_scheduling_split() is a self-scheduling function, so it not only updates itself if the underlying variable changes (e.g., `integer_age` changes) but also ensures that the underlying variable is updated at the scheduled moment. This is useful when using continuous variables like age or time, as there may not be another event happening at the time the split is supposed to happen. The above table could also be written as:

```

table Person TabNumberMoves //EN Internal migration rate
[WITHIN(SIM_YEAR_RANGE, calendar_year)]
{
    {
        number_moves / duration() //EN Migration rate decimals=4
    }
    * self_scheduling_split(age, AGE5_PART) //EN Age group
};

```

Note that by producing their own events, self-scheduling states are resource intensive and unnecessary use should be avoided, including multiple use of the same split function, e.g., in various tables.

Converting probabilities to rates

Probabilities and rates are linked through the concept of survival, which is the probability that no event happens within a period where an actor is at a constant risk of an event.

```

within one time interval:

survival = exp(-rate)    the probability that nothing happens under a constant risk
'rate'
rate = -ln(survival)    the rate, at which survival % of the population does not
experience an event

```

Note that for a constant rate, the waiting time of an event is exponentially distributed, while the number of events follows the Poisson distribution.

7.6.3. How to reproduce this modeling step

All code, except for tables, is added at this step within the new module, `Migration.mpp`. The module has

a typical organization:

- Documentation
- Types
- Parameters
- Actor definitions
- Event implementation

The module Migration.mpp

The following describes how to add a module Migration.mpp.

Documentation

```
//LABEL(Migration, EN) Migration

/* NOTE(Migration, EN)

    Added at Step 5

    This module implements internal migration. It is based on age and sex specific
    transition matrices. Only one transition per year is allowed. The module has three
    parameters, one to switch migration on/off, the second for the probability to move away (by
    province, age and sex), the third to sample the destination province (by origin, age and sex).
*/
```

Types

The model uses five-year age groups, which can built by a partition.

```
partition AGE5_PART { 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60 }; //EN Age
Groups
```

Parameters

The age partition into five-year age groups is used as a dimension type of the two parameters handling migration: the probability to leave and the destination. As the destination is sampled, a cumrate parameter is used.

```
parameters
{
    logical ModelMigration; //EN
    Migration switched on/off
    double MigrationProbability[SEX][AGE5_PART][PROVINCE_NAT]; //EN
    Migration probability
    cumrate MigrationDestination[SEX][PROVINCE_NAT][AGE5_PART][PROVINCE_NAT]; //EN
    Migration Destination
};

parameter_group PG_Migration //EN
Internal migration
{
    ModelMigration, MigrationProbability, MigrationDestination
};
```

Actor block

The module has two simple states: the first records the number of moves of a person, the second records the age at the last move. (Only one move is allowed by age year.) There is one event, the migration event, at which a new province is chosen.

```
actor Person
{
    integer number_moves = { 0 };           //EN Number of
interprovincial moves
    logical age_at_last_move = { 999 };     //EN Age at last migration
    event    timeMigrationEvent, MigrationEvent; //EN Migration Event
};
```

Event implementation

The event time for an inter-provincial move is modeled in continuous time. To do so, the given probability is transformed into a hazard rate = $-\ln(1 - \text{probability})$.

```
TIME Person::timeMigrationEvent()
{
    // get the current age index using SPLIT()
    int nAge5 = SPLIT(integer_age, AGE5_PART);

    // get the probability to move
    double dMoveProb = MigrationProbability[sex][nAge5][province_nat];

    // Check if a person is at risk for moving
    if (ModelMigration && in_projected_time && !has_moved_this_year && dMoveProb >
0.0)
    {
        if (dMoveProb >= 1.0) return WAIT(0); // if there is a 100% probability move
immediately
        else // there is a positive probability 100%
        {
            // calculate a random waiting time based on the given probability
converted to a hazard
            // rate = -log(1-probability)
            return WAIT(-log(RandUniform(2)) / -log(1 - dMoveProb));
        }
    }
    return TIME_INFINITE;
}

void Person::MigrationEvent()
{
    int nDestination;
    int nAge5 = SPLIT(integer_age, AGE5_PART);

    // Sample the destination
    Lookup_MigrationDestination(RandUniform(3), sex, province_nat, nAge5,
&nDestination);

    // move the actor to the destination
    province_nat = (PROVINCE_NAT)nDestination;

    // update indicators
    has_moved_this_year = TRUE;
    number_moves++;
}
```

```
}

```

New tables in Tables.mpp

We add two tables, one for the average population by sex, province, and projected year, and the other for simulated migration rates by sex, age group, and province of origin.

```
table Person PopulationTableProvAverage //EN Average Population by province
[WITHIN(SIM_YEAR_RANGE, calendar_year)]
{
  sex + *
  {
    duration() //EN Average Population
  }
  * province_nat +
  * sim_year
};

table Person TabNumberMoves //EN Internal migration rate
[WITHIN(SIM_YEAR_RANGE, calendar_year)]
{
  sex + *
  {
    number_moves / duration() //EN Migration rate decimals=4
  }
  * split(integer_age, AGE5_PART) //EN Age group
  * province_nat +
};
```

7.6.4. Model versions

Converting the model to a time-based and/or micro-data-based version

As in any modeling step in this guide, the case-based model can be converted to time-based by replacing the DYNAMIS-POP-MRT.mpp module (see Appendix for code) and adapting its settings in ModelSettings.mpp.

As in any step starting from Step 1, the starting population type can be converted from a sampling table to a micro-data file by replacing the starting population file (see Appendix for code) and adapting the settings in ModelSettings.mpp.

No changes affecting model conversions are made at this step, compared to the previous step. Note that model conversion is discussed in detail in *Step 11: Conversion to a Time-Based Model*.

7.7. Step 7: Emigration

7.7.1. Overview

This step adds emigration to the model. Emigration is based on emigration rates by age group, sex, and province of residence. The module can be switched on/off by the user.

7.7.2. Concepts

This module adds emigration to the model. It does not introduce any new Modgen concept, but can be seen as a typical Modgen module containing most basic concepts.

7.7.3. How to reproduce this modeling step

Add emigration to the model by using all new code (except for tables), which are within the new module Emigration.mpp, which follows a typical outline.

The Emigration.mpp module

Add a new module Emigration.mpp and its documentation.

```
//LABEL(Emigration, EN) Emigration

/* NOTE(Emigration, EN)

    Added at Step 6

    This module implements emigration based on emigration rates by age group, sex,
    and
    province of residence. The module can be switched on/off by the user.
*/
```

Parameters

```
parameters
{
    logical ModelEmigration; //EN Switch
    emigration on/off
    double EmigrationRates[SEX][AGE5_PART][PROVINCE_NAT]; //EN Emigration
    Rates
};

parameter_group PG_Emigration //EN Emigration
{
    ModelEmigration, EmigrationRates
};
```

Actor definitions

```
actor Person
{
    logical has_emigrated = { FALSE }; //EN Person has
    emigrated
    event timeEmigrationEvent, EmigrationEvent; //EN Emigration
};
```

Event implementation

```
TIME Person::timeEmigrationEvent ()
{
    if (ModelEmigration && calendar_year >= MIN(SIM_YEAR_RANGE) &&
        EmigrationRates[sex][SPLIT(integer_age, AGE5_PART)][province_nat] > 0.0)
    {
        return WAIT(-log(RandUniform(10)) / EmigrationRates[sex][SPLIT(integer_age,
        AGE5_PART)][province_nat]);
    }
    else return TIME_INFINITE;
}

void Person::EmigrationEvent ()
```

```
{
  has_emigrated = TRUE;
  Finish();
}
```

Tables

We add a validation table for emigration rates.

```
table Person EmigrationRatesTable //EN Emigration
rates
[WITHIN(SIM_YEAR_RANGE, calendar_year)]
{
  sex + *
  {
    transitions(has_emigrated, FALSE, TRUE) / duration() //EN Emigration
rates decimals=4
  }
  *split(integer_age, AGE5_PART) //EN Age group
  * province_nat +
};
```

Once again, the table for remaining life expectancy failed, as emigrants are recorded only while they are in the country. To fix this, we produce the table at death, and therefore exclude all who emigrated and whose death is not recorded.

```
table Person RemainingLifeExpectancyAge //EN Remaining life
expectancy by age at start
[trigger_entrances(alive,FALSE) && in_projected_time && time_of_birth <
MIN(SIM_YEAR_RANGE)]
{
  tab_age_start *
  {
    (value_in(age) - value_in(tab_age_start)) / unit //EN Remaining life
expectancy in simulation decimals=4
  }
};
```

7.7.4. Model versions

Converting the model to a time-based and/or micro-data-based version

As in any modeling step in this guide, the case-based model can be converted to time-based by replacing the DYNAMIS-POP-MRT.mpp module (see Appendix for code) and adapting its settings in ModelSettings.mpp.

As in any step starting from Step 1, the starting population type can be converted from a sampling table to a micro-data file by replacing the starting population file (see Appendix for code) and adapting the settings in ModelSettings.mpp.

No changes affecting model conversions are made at this step, compared to the previous step. Note that model conversion is discussed in detail in *Step 11: Conversion to a Time-Based Model*.

7.8. Step 8: Immigration

7.8.1. Overview

This step adds immigration to the model. Immigration is based on immigration numbers by age, sex, and destination province, and can be switched on and off. Model parameters are the number and sex of immigrants by year, the age distribution by sex, and the distribution of destination provinces by sex and age. Immigrants are created at the moment of immigration with the age at immigration.

This step also adds population scaling to the model. The number of persons by age, sex, and province in the distributional table for the starting population and the parameter for the number of future immigrants is used to calculate the “true” population size, to which the model output can be scaled automatically if “scale population” is selected in the model settings. The output then reflects the real population size, independent of the chosen sample size.

7.8.2. Concepts

This module adds immigration to the model. One new concept totals the values in population tables and copies them into a model-generated cumrate table in the pre-simulation phase. This is necessary, as cumrate parameters are internally optimized for sampling, but lose information about their original cell values and their sum.

Another concept introduced at this step is automatic population scaling. If the user clicks “population scaling” in the model settings, the model scales automatically to the population size recorded in the starting population and immigrant tables. This overwrites any other scaling target.

Copying parameters from double to cumrate tables in pre-simulation

The following code loops through a population table, adding the cell values and copying values to a cumrate parameter of the same shape.

```
// Count and copy Starting population
StartPopSize = 0.0;
for (int nSex = 0; nSex < SIZE(SEX); nSex++)
{
    for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
    {
        for (int nProv = 0; nProv < SIZE(PROVINCE_NAT); nProv++)
        {
            StartPopSize = StartPopSize + StartingPopulation[nSex][nAge][nProv];
            StartingPopulationTable[nSex][nAge][nProv] =
StartingPopulation[nSex][nAge][nProv];
        }
    }
}
```

Population scaling

Case-based models

Case-based models provide an option to scale model output to a target population size in the Scenario, where users can specify target population size. Target population size corresponds to the number of cases created by the simulation engine, i.e., the generation zero (who then can have children, grand-children, etc., who are not counted here). As the target population can include persons born in the future (e.g., immigrants), it is not a readily available number. Modgen allows overwriting this number with a number calculated in the simulation, e.g., adding together all values from the population table and the table of

immigrants.

The following code gives three scaling options:

- No scaling: a parameter “scale population” and the scenario settings for scaling are not clicked. The model produces the number of actors, as in the scenario settings’ number of cases.
- Scaling as in scenario settings: a parameter “scale population” is not clicked, but scaling is switched on in the scenario settings. The population is scaled to the number in the settings.
- Automatic scaling to the “true” population size is determined in the simulation. The parameter for automatic population scaling is clicked, and the settings of the scenario options are overwritten.

The following code uses the following parameters, settings, and functions:

- ScalePopulation: a model parameter. The code is effective only if it is true; otherwise scaling is handled by scenario settings.
- SetPopulation(TotalPopSize) overwrites the value of the target size in the scenario settings. If scaling is clicked on in the scenario settings, scaling is handled automatically.
- GetPopulationScalingRequired() returns true if the user has clicked scaling in the scenario settings. If not, code for weighting the actors has to be added, as it will not be handled automatically.
- Set_actor_weight() and Set_actor_subsample_weight() are the functions used for setting weights.
- TotalPopSampleSize is a model_generated parameter identical to the number of simulated cases specified in the scenario settings.
- TotalPopSize is the total target population size calculated in the model from parameter tables.

```
void Person::Start(long PersonNr, Person *peMother)
{
    if (ScalePopulation)
    {
        SetPopulation(TotalPopSize);

        // Set the actor weight if population scaling not clicked in Scenario
settings
        if (!GetPopulationScalingRequired())
        {
            Set_actor_weight(TotalPopSize / TotalPopSampleSize);
            Set_actor_subsample_weight(TotalPopSize / TotalPopSampleSize);
        }
        [...]
    }
}
```

Note that, if desired, models are always scaled to TotalPopSize if population scaling is turned on in the scenario settings. No additional parameter for automatic scaling is required and the target population size is always overwritten by the calculated one. In this case, weighting can be done with one line of code, which can also be placed into the pre-simulation function after TotalPopSize is calculated there. This approach works only in case-based models.

```
if (GetPopulationScalingRequired()) SetPopulation(TotalPopSize);
```

Time-based models

There are no scenario settings for population size and scaling in time-based models, where the population size has to be determined by a parameter. Unlike case-based models, TotalPopSampleSize is a normal parameter that must be set by the user.

The code for time-based models is:

```
void Person::Start(Person *peMother)
{
    if (ScalePopulation)
    {
        Set_actor_weight(TotalPopSize / TotalPopSampleSize);
        Set_actor_subsample_weight(TotalPopSize / TotalPopSampleSize);
    }
    [...]
}
```

7.8.3. How to reproduce this modeling step

At this step, we add immigration to the model by adding a new module, `Immigration.mpp`, which has many similarities to the `StartPopSampling` module, as its only actor function is used to sample the characteristics of new immigrants. The sampling function is called at `Start()`, which has to be adapted to deal with the new type of persons besides those in the starting population and simulated births. Also, because two populations—the starting population and population of immigrants—are created by the simulation engine, we must know the relation between the two population sizes. Counting the population parameter tables is performed in the Pre-Simulation. This information will be used to automatically scale the model output.

The `Immigration.mpp` module

Create and document

Add a new module `Immigration.mpp` and its documentation:

```
//LABEL(Immigration, EN) Immigration

/* NOTE(Immigration, EN)

    Added at Step 7

    This module implements immigration. Immigration can be switched on and off.
    Model parameters are the number and sex of immigrants by year, the age
    distribution by sex,
    and the distribution of destination provinces by sex and age.
    Immigrants are created at the moment of immigration with the age at immigration
*/
```

Parameters

```
parameters
{
    logical ModelImmigration; //EN
    Switch immigration on/off
    double NumberImmigrants[SIM_YEAR_RANGE][SEX]; //EN
    Number of immigrants
    model_generated cumrate[2] NumberImmigrantsTable[SIM_YEAR_RANGE][SEX]; //EN
    Number of immigrants (sampling)
    cumrate AgeImmigrants[SEX][AGE_RANGE]; //EN Age
    distribution of immigrants
    cumrate DestinationImmigrants[SEX][AGE5_PART][PROVINCE_NAT]; //EN
    Destination of immigrants
};

parameter_group PG_Immigration //EN
```

```

Immigration
{
    ModelImmigration, NumberImmigrants, AgeImmigrants, DestinationImmigrants
};

```

Actor states and function

```

actor Person
{
    TIME    time_of_immigration = { TIME_INFINITE };    //EN Time of immigration
    void    GetImmigrantCharacteristics();             //EN Sample immigrant
    characteristics
};

```

Function implementation

```

::

```

```

void Person::GetImmigrantCharacteristics() {
    // Sex and time of immigration int nSex, nYear; Lookup_NumberImmigrantsTable(RandUniform(11), &nYear, &nSex); sex = (SEX)nSex; time_of_immigration = MIN(SIM_YEAR_RANGE) + nYear + RandUniform(12);
    // Time of birth int nAge; Lookup_AgeImmigrants(RandUniform(14), sex, &nAge); double dAge = nAge + RandUniform(13); time_of_birth = time_of_immigration - dAge;
    // Province of immigration int nProvince; Lookup_DestinationImmigrants(RandUniform(9), sex, SPLIT(nAge, AGE5_PART), &nProvince); province_nat = (PROVINCE_NAT)nProvince;
}

```

Changes in PersonCore.mpp

New classification and states

By adding immigrants to the population, we now have three types of persons: those from the starting population, those born in the simulation, and those entering the country as immigrants during the simulation. We add a new state, recording this information.

```

classification PERSON_TYPE //EN Person Type
{
    PT_START, //EN Person from
    Starting Population
    PT_CHILD, //EN Person born in
    simulation
    PT_IMMIGRANT //EN Immigrant
};

actor Person
{
    PERSON_TYPE    person_type = { PT_START }; //EN Person type
    [...]
};

```

Changes in ModelSettings.mpp

Model settings are extended to include immigrants and count the population size of future immigrants.

```

parameters
{
    model_generated long    StartPopSampleSize; //EN Sample size of starting

```

```

population
    model_generated double   StartPopSize;           //EN Total population size

    model_generated long     ImmiPopSampleSize;      //EN Sample size of starting
population
    model_generated double   ImmiPopSize;           //EN Total population size

    model_generated long     TotalPopSampleSize;     //EN Sample size of starting
population
    model_generated double   TotalPopSize;          //EN Total population size

    model_generated logical  ScalePopulation;       //EN Scale population
    model_generated double   ActorWeight;          //EN Actor Weight
};

void PreSimulation()
{
    // Starting population
    StartPopSize = 0.0;
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
        {
            for (int nProv = 0; nProv < SIZE(PROVINCE_NAT); nProv++)
            {
                StartPopSize = StartPopSize + StartingPopulation[nSex][nAge][nProv];
                StartingPopulationTable[nSex][nAge][nProv] =
StartingPopulation[nSex][nAge][nProv];
            }
        }
    }

    // Immigrant population
    ImmiPopSize = 0.0;
    if (ModelImmigration) // If immigration is switched on
    {
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
            {
                ImmiPopSize = ImmiPopSize + NumberImmigrants[nYear][nSex];
                NumberImmigrantsTable[nYear][nSex] = NumberImmigrants[nYear][nSex];
            }
        }
    }

    //Total population
    TotalPopSize = ImmiPopSize + StartPopSize;
    TotalPopSampleSize = GetAllCases();

    // Weight
    ActorWeight = TotalPopSize / TotalPopSampleSize;

    StartPopSampleSize = (long)(StartPopSize / ActorWeight);
    ImmiPopSampleSize = TotalPopSampleSize - StartPopSampleSize;

    //Scale Population
    ScalePopulation = GetPopulationScalingRequired();
    if (ScalePopulation && model_is_case_based)

```

```

    {
        SetPopulation(TotalPopSize);
    }
};

```

Changes in the Start() function - 3 types of persons and population scaling

The Start() function is now rearranged, to account for the three types of persons. Note that new immigrants are created at the moment of immigration and not at their date of birth, so they are created at their age at immigration.

Also, information on total population size is used now to scale the model output to the true population size if requested. If a user selects population scaling in the parameters, the scenario settings are ignored and the SetPopulation() (over)writes the target value of population scaling in the user settings. As the scenario settings for time-based models do not include the scaling option, the code checks if the model is time or case-based. If case-based, it overwrites the scenario settings if the parameter for automatic scaling to the true population is switched on.

```

void Person::Start(long PersonNr, Person *peMother)
{
    // Population scaling
    if (ScalePopulation)
    {
        if (model_is_case_based)
        {
            SetPopulation(TotalPopSize);

            // Set the actor weight if population scaling not clicked in Scenario
settings
            if (!GetPopulationScalingRequired())
            {
                Set_actor_weight(TotalPopSize / TotalPopSampleSize);
                Set_actor_subsample_weight(TotalPopSize / TotalPopSampleSize);
            }
        }
        else
        {
            Set_actor_weight(TotalPopSize / TotalPopSampleSize);
            Set_actor_subsample_weight(TotalPopSize / TotalPopSampleSize);
        }
    }

    // Determine the person type
    if (peMother != NULL) //
Born in simulation
    {
        person_type = PT_CHILD;
    }
    else if (PersonNr < TotalPopSampleSize * (StartPopSize / TotalPopSize)) //
Person from start population
    {
        person_type = PT_START;
    }
    else //
Immigrant
    {
        person_type = PT_IMMIGRANT;
    }
}

```

```

// Initializations person from start population
if (person_type == PT_START)
{
    GetStartCharacteristics();
    age = 0;
    time = time_of_birth;
}
// Initializations persons born in simulation
else if (person_type == PT_CHILD)
{
    // assign a sex according to the sex ratio parameter
    sex = MALE;
    if (RandUniform(5) < 100.0 / (100.0 + SexRatio[RANGE_POS(SIM_YEAR_RANGE,
calendar_year)])) sex = FEMALE;

    // inherit characteristics from the mother
    time_of_birth = peMother->time;          // The time of creation
    province_nat = peMother->province_nat;    // The province at birth
    age = 0;
    time = time_of_birth;
}
// Initializations immigrants
else
{
    GetImmigrantCharacteristics();
    age = time_of_immigration - time_of_birth;
    time = time_of_immigration;
}
}

```

New tables

We add three tables to the group of migration tables, which can also be used to validate the results against the parameters.

```

table_group TG_04_MigrationTables //EN Migration
{
    TabNumberMoves, EmigrationRatesTable,
    NumberImmigrantsTab, AgeImmigrantsTable,
    DestinationImmigrantsTable
};

table Person NumberImmigrantsTab //EN Number of
immigrants
[person_type == PT_IMMIGRANT && WITHIN(SIM_YEAR_RANGE, calendar_year)]
{
    {
        entrances(set_alive, TRUE) //EN
    }
    Immigrants
}
* sim_year
* sex +
};

table Person AgeImmigrantsTable //EN Age at
immigration
[person_type == PT_IMMIGRANT && WITHIN(SIM_YEAR_RANGE, calendar_year) &&
trigger_entrances(set_alive, TRUE)]
{

```

```

    {
        unit //EN Immigrants
    }
    * integer_age
    * sex +
};

table Person DestinationImmigrantsTable //EN Destination
of immigrants
[person_type == PT_IMMIGRANT && WITHIN(SIM_YEAR_RANGE, calendar_year) &&
trigger_entrances(set_alive, TRUE)]
{
    sex+ *
    {
        unit //EN Immigrants
    }
    * split(integer_age, AGE5_PART) //EN Age group
    * province_nat +
};

```

Not that all new tables filter to immigrants. After adding immigrants, we have to revise the existing tables if they are affected by the new type of persons. This is the case where we count the number of births by counting “set_alive” transition: we must include immigrants, who are set alive at the age of immigration.

```

table Person NumberBirths //EN Number of births
[WITHIN(SIM_YEAR_RANGE, calendar_year) && person_type == PT_CHILD ]
{
    {
        entrances(set_alive, TRUE) //EN Births
    }
    * sim_year
    * sex +
};

```

The second table affected by the changes is the remaining life expectancy of the people in the starting population. Filtering to this person type explicitly solves the problem.

```

table Person RemainingLifeExpectancyAge //EN Remaining life
expectancy by age at start
[trigger_entrances(alive, FALSE) && person_type == PT_START]
{
    tab_age_start *
    {
        (value_in(age) - value_in(tab_age_start)) / unit //EN Remaining life
        expectancy in simulation decimals=4
    }
};

```

7.8.4. Model versions

Converting the model to a time-based and/or micro-data-based version

As in any modeling step in this guide, the case-based model can be converted to time-based by replacing the DYNAMIS-POP-MRT.mpp module (see Appendix for code) and adapting its settings in ModelSettings.mpp.

As in any step starting from Step 1, the starting population type can be converted from a sampling table to a micro-data file by replacing the starting population file (see Appendix for code) and adapting the

settings in ModelSettings.mpp.

Note that model conversion is discussed in detail in *Step 11: Conversion to a Time-Based Model*.

The module ModelSettings.mpp in alternative versions of Step 7

Sampling - time based

```

parameters
{
    model_generated long      StartPopSampleSize;          //EN Sample size of starting
population
    model_generated double    StartPopSize;                //EN Total population size

    long                      TotalPopSampleSize;          //EN Sample size of starting
population
    model_generated double    TotalPopSize;                //EN Total population size

    model_generated long      ImmiPopSampleSize;          //EN Sample size of starting
population
    model_generated double    ImmiPopSize;                //EN Total population size

    logical                   ScalePopulation;             //EN Scale population
    model_generated double    ActorWeight;                //EN Actor Weight
};

parameter_group PG00_ModelSettings                        //EN Model Settings
{
    TotalPopSampleSize, ScalePopulation
};

void PreSimulation()
{
    // Starting population
    StartPopSize = 0.0;
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
        {
            for (int nProv = 0; nProv < SIZE(PROVINCE_NAT); nProv++)
            {
                StartPopSize = StartPopSize + StartingPopulation[nSex][nAge][nProv];
                StartingPopulationTable[nSex][nAge][nProv] =
StartingPopulation[nSex][nAge][nProv];
            }
        }
    }

    // Immigrant population
    ImmiPopSize = 0.0;
    if (ModelImmigration) // If immigration is switched on
    {
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
            {
                ImmiPopSize = ImmiPopSize + NumberImmigrants[nYear][nSex];
                NumberImmigrantsTable[nYear][nSex] = NumberImmigrants[nYear][nSex];
            }
        }
    }
}

```

```

//Total population
TotalPopSize = StartPopSize + ImmiPopSize;
StartPopSampleSize = TotalPopSampleSize * (StartPopSize / TotalPopSize);

// Weight
ActorWeight = TotalPopSize / TotalPopSampleSize;

//Scale Population
if (ScalePopulation && model_is_case_based)
{
    SetPopulation(TotalPopSize);
}
};

```

Micro data file - case based

```

parameters
{
    model_generated long    StartPopSampleSize;           //EN Sample size of starting
population
    double                StartPopSize;                   //EN Total population size

    model_generated long    ImmiPopSampleSize;           //EN Sample size of starting
population
    model_generated double  ImmiPopSize;                 //EN Total population size

    model_generated long    TotalPopSampleSize;         //EN Sample size of starting
population
    model_generated double  TotalPopSize;               //EN Total population size

    model_generated logical ScalePopulation;             //EN Scale population
    model_generated double  ActorWeight;                 //EN Actor Weight
};

parameter_group PG_ModelSettings //EN Model Settings
{
    StartPopSize
};

void PreSimulation()
{
    // Immigrant population
    ImmiPopSize = 0.0;
    if (ModelImmigration) // If immigration is switched on
    {
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
            {
                ImmiPopSize = ImmiPopSize + NumberImmigrants[nYear][nSex];
                NumberImmigrantsTable[nYear][nSex] = NumberImmigrants[nYear][nSex];
            }
        }
    }

    //Total population
    TotalPopSize = ImmiPopSize + StartPopSize;
    TotalPopSampleSize = GetAllCases();
}

```

```

// Weight
ActorWeight = TotalPopSize / TotalPopSampleSize;

StartPopSampleSize = (long)(StartPopSize / ActorWeight);
ImmiPopSampleSize = TotalPopSampleSize - StartPopSampleSize;

//Scale Population
ScalePopulation = GetPopulationScalingRequired();
if (ScalePopulation && model_is_case_based)
{
    SetPopulation(TotalPopSize);
}
};

```

Micro data file - time based

```

parameters
{
    model_generated long    StartPopSampleSize;    //EN Sample size of starting
population
    double                StartPopSize;           //EN Total population size

    model_generated long    ImmiPopSampleSize;    //EN Sample size of starting
population
    model_generated double  ImmiPopSize;          //EN Total population size

    long                  TotalPopSampleSize;    //EN Sample size of starting
population
    model_generated double  TotalPopSize;        //EN Total population size

    logical                ScalePopulation;       //EN Scale population
    model_generated double  ActorWeight;         //EN Actor Weight
};

parameter_group PG_ModelSettings //EN Model Settings
{
    StartPopSize, TotalPopSampleSize, ScalePopulation
};

void PreSimulation()
{
    // Immigrant population
    ImmiPopSize = 0.0;
    if (ModelImmigration) // If immigration is switched on
    {
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
            {
                ImmiPopSize = ImmiPopSize + NumberImmigrants[nYear][nSex];
                NumberImmigrantsTable[nYear][nSex] = NumberImmigrants[nYear][nSex];
            }
        }
    }

    //Total population
    TotalPopSize = ImmiPopSize + StartPopSize;

    // Weight
    ActorWeight = TotalPopSize / TotalPopSampleSize;
}

```

```

StartPopSampleSize = (long) (StartPopSize / ActorWeight);
ImmiPopSampleSize = TotalPopSampleSize - StartPopSampleSize;

//Scale Population
if (ScalePopulation && model_is_case_based)
{
    SetPopulation(TotalPopSize);
}
};

```

7.9. Step 9: Micro-Data Output

7.9.1. Overview

This step adds micro-data output to the model, allowing the user to set a time when a micro-data file is written. All code is within the new module, `MicroDataOutput.mpp`, which can be added at any step of model development.

This step completes the micro-simulation implementation of a typical macro population projection model.

7.9.2. Concepts

Modgen functionality for micro-data output

Modgen includes some functionality to produce micro-data output in the form of CSV files.

For CSV output files, the file has to be declared and a parameter for the file name must be created:

```

output_csv out_csv; //EN Microdata output csv object

parameters
{
    file    MicroRecordFileName; //EN File name micro-data output file
};

```

The output file has to be opened before and closed after the simulation. This can be at the beginning and the end of the `Simulation()` function, or in the pre- and post- simulation.

```

// Open the output file
out_csv.open(MicroRecordFileName);

// Close the output file
out_csv.close();

```

Output records can be produced within a function or event:

```

// Push the fields into the output record.
out_csv << time_of_birth;
out_csv << (int)sex;
out_csv << (int)province_nat;

// All fields have been pushed, now write the record.
out_csv.write_record();

```

7.9.3. How to reproduce this modeling step

The full code for micro-population is contained in one file, which can be added at any point in model development. Output variables can also be edited and added at any point in model development.

The MicroDataOutput.mpp module

The following code is the full module:

```
// LABEL(MicroDataOutput, EN) Micro data output
/* NOTE(MicroDataOutput, EN)
    Added at Step 8: This module can be added at any step of model development
    This module implements micro data output written to a csv file.
    Users can specify the time at which a micro-data file is written out and choose a
    file name.
    The module can be switched on and off.
*/
output_csv out_csv; //EN Microdata output
csv object

parameters
{
    logical WriteMicrodata; //EN Write micro-data
    output file Y/N
    double TimeMicroOutput; //EN Time of
    micro-data output
    file MicroRecordFileName; //EN File name
    micro-data output file
};

parameter_group PG05_Files //EN Microdata output
{
    WriteMicrodata, MicroRecordFileName, TimeMicroOutput
};

actor Person
{
    TIME time_microdata_output = { TIME_INFINITE }; //EN Time for
    microdata output
    void WriteMicroRecord_Start(); //EN Initialization
    for microdata output event
    hook WriteMicroRecord_Start, Start;
    event timeWriteMicroRecord, WriteMicroRecord; //EN Write micro-data
    record event
};

void Person::WriteMicroRecord_Start()
{
    if (WriteMicrodata && TimeMicroOutput >= time) time_microdata_output =
    TimeMicroOutput;
    else time_microdata_output = TIME_INFINITE;
}

TIME Person::timeWriteMicroRecord()
{
```

```

    return time_microdata_output;
}

void Person::WriteMicroRecord()
{
    // Push the fields into the output record.
    out_csv << time_of_birth;
    out_csv << (int)sex;
    out_csv << (int)province_nat;

    // All fields have been pushed, now write the record.
    out_csv.write_record();

    // do only once
    time_microdata_output = TIME_INFINITE;
}

void PreSimulation()
{
    if (WriteMicrodata)    if (WriteMicrodata) { out_csv.open(MicroRecordFileName);
}
}

void PostSimulation()
{
    if (WriteMicrodata) { out_csv.close(); }
}

```

7.9.4. Model versions

Converting the model to a time-based and/or micro-data-based version

As in any modeling step in this guide, the case-based model can be converted to time-based by replacing the DYNAMIS-POP-MRT.mpp module (see Appendix for code) and adapting its settings in ModelSettings.mpp.

As in any step starting from Step 1, the starting population type can be converted from a sampling table to a micro-data file by replacing the starting population file (see Appendix for code) and adapting the settings in ModelSettings.mpp.

No changes affecting model conversions are made at this step, compared to the previous step. Note that model conversion is discussed in detail in *Step 11: Conversion to a Time-Based Model*.

7.10. Step 10: Micro-Data Input

7.10.1. Overview

In this step, we convert the starting population type of the model. Instead of sampling characteristics from a distributional table, we read in a micro-data file.

7.10.2. Concepts

Modgen functionality for micro-data input

Modgen includes some functionality to produce micro-data input in form of CSV files.

For CSV input files, the file has to be declared and a parameter for the file name must be created:

```
input_csv in_csv; //EN Microdata input csv object

parameters
{
    file    MicroDataInputFile; //EN File name of starting population
};
```

The fields of a record can be addressed by an index starting at 0, which is best handled creating a classification of field names:

```
classification PERSON_MICRODATA_COLUMNS //EN fields in the microdata input
file
{
    PMC_BIRTH, //EN Time of birth
    PMC_SEX, //EN Sex
    PMC_PROVINCE //EN Province
};
```

The input file has to be opened before and closed after the simulation. This can be at the beginning and the end of the Simulation() function, or in the pre- and post- simulation.

```
// Open the input file
in_csv.open(MicroDataInputFile);

// Close the input file
in_csv.close();
```

Input records can be read within a read function or event by a function that takes one argument, which is the line number of the record:

```
in_csv.read_record(i_lLine);
```

Once a record is read, its fields can be accessed by the index as defined in our classification:

```
time_of_birth = in_csv[PMC_BIRTH]; // time of birth
if ((int)in_csv[PMC_SEX] == 1) sex = MALE; else sex = FEMALE; // sex
province_nat = (PROVINCE_NAT)(int)in_csv[PMC_PROVINCE]; // province
```

Note that all values of the CSV file are read in as double and have to be casted to other types if required. In C++, this is done by prompting the variable with (NewType).

7.10.3. How to reproduce this modeling step

The full code for micro-population input is contained in one file—StartPopFile.mpp—which replaces the file StartPopSampling.mpp. This replacement of files and model type can also be performed at any previous step of model development. Besides this file replacement, a few adaptations in ModelSettings.mpp have to be made.

The StartPopFile.mpp module

The following code is the full module:

```
//LABEL(StartPopFile, EN) Starting population File

/* NOTE(StartPopFile, EN)

Added at step 9 replacing StartPopSampling.mpp
```

This replacement can be made at **any** previous modeling step

This file replaces StartPopSampling.mpp **and** converts the model **from a** model that samples starting characteristics to a model that reads **in** a micro-population file. Variables are time of birth, sex, **and** region

```

*/

input_csv in_csv; //EN Microdata input csv
object

classification PERSON_MICRODATA_COLUMNS //EN fields in the microdata
input file
{
    PMC_BIRTH, //EN Time of birth
    PMC_SEX, //EN Sex
    PMC_PROVINCE //EN Province
};

parameters
{
    file MicroDataInputFile; //EN File name of starting
population
};

parameter_group PG00_MicroDataInputFile //EN Starting Population file
{
    MicroDataInputFile
};

actor Person
{
    void GetStartCharacteristics(long i_lLine); //EN Read characteristics at
birth
};

void Person::GetStartCharacteristics(long i_lLine)
{
    // Read record
    in_csv.read_record(i_lLine);

    // Assigns values from record
    time_of_birth = in_csv[PMC_BIRTH]; // time of
birth
    if ((int)in_csv[PMC_SEX] == 1) sex = MALE; else sex = FEMALE; // sex
    province_nat = (PROVINCE_NAT) (int)in_csv[PMC_PROVINCE]; // province
}

void PreSimulation()
{
    in_csv.open(MicroDataInputFile);
}

void PostSimulation()
{
    in_csv.close();
}

```

New settings in ModelSettings.mpp

Parameters: The only change in parameters is that the size of the starting population is changed from a model-generated parameter to a normal double parameter, as the model now cannot calculate the size of the starting population automatically.

```
parameters
{
    [...]
    double StartPopSize;           //EN Starting population size
    [...]
};

parameter_group PG00_ModelSettings //EN Model Settings
{
    StartPopSize
};
```

In the `PreSimulation()` function, we accordingly have to remove the loop, which previously calculated the starting population size. This is the only required change in settings.

Below the full `ModelSettings.mpp` file.

The microdata file

At this point, we need a micro-data file in CSV format with the fields.

```
- Birth date:   e.g. 1966.158
- Sex:         0 - Female, 1 - Male
- Province:    0 - 12
```

Such a file can be generated in the previous model version, which allows writing a micro-data file of this format. As model development moves beyond these three characteristics, such a file can be produced by statistical software.

7.11. Step 11: Conversion to a Time-Based Model

7.11.1. Overview

In this step, we convert to a time-based model, which will be the starting point for the second stage of model development, allowing for a broad range of additional features, like actor interactions and automatic model alignment to external targets.

7.11.2. Concepts

Time-Based Models

In time-based models, all persons are simulated simultaneously through time. While more resource intensive, such models allow for communication between all persons or any other entities in the simulation.

Technically, any case-based model can be implemented as a time-based model. Such a conversion can be done easily by replacing the simulation engine file—in our case the `DYNAMIS-POP-MRT.mpp` file—by an engine for time-based models. A template for such a file is provided here. Developers typically do not touch the code of the simulation engine file, so the same file can be used in all modeling steps.

Time-Based models have different scenario-setting parameters in the graphical user interface.

- A new parameter for the end of the simulation time. At this point in time the simulation is stopped automatically.
- Three parameters used so far are missing: the sample size, the total population size, and the box to check for automatic population scaling. If required, these parameters must be added as normal model parameters.
- Time-Based models do not allow for sub-samples, as all actors have to be able to communicate with each other. Instead, for the calculation of monte carlo variability in outputs, the user can define a number of model replicates created and Modgen automatically averages over the outcome results.

At this modeling step, we do not add any functionality to the model and accordingly do not make use yet of the added modeling possibilities.

7.11.3. How to reproduce this modeling step

To convert the model from case based to time based, the code in the simulation engine file has to be replaced. The code for a time-based model can be found in the Appendix.

New settings in ModelSettings.mpp

Parameters: As the user interface of the scenario settings of time-based models do not include parameters for sample and population sizes as well as population scaling, these parameters are changed into normal model parameters. This gives us added flexibility as to which population size parameters can be set by the user, and which to calculate based on these settings. We will use this flexibility, to let the user choose the size of the starting population file and the corresponding real size of the starting population. Based on the sampling table for immigrants, the sample and total size of the immigrant population, and also of the total population can be calculated automatically.

```
parameters
{
    long                StartPopSampleSize;           //EN Sample size of starting
population
    double              StartPopSize;                 //EN Total population size

    model_generated long ImmiPopSampleSize;           //EN Sample size of starting
population
    model_generated double ImmiPopSize;               //EN Total population size

    model_generated long TotalPopSampleSize;          //EN Sample size of starting
population
    model_generated double TotalPopSize;              //EN Total population size

    logical             ScalePopulation;              //EN Scale population
    model_generated double ActorWeight;              //EN Actor Weight
};

parameter_group PG_ModelSettings //EN Model Settings
{
    StartPopSize, StartPopSampleSize, ScalePopulation
};
```

In the **PreSimulation()** function, we accordingly have to change how and which parameters are calculated. The actor weight can now be calculated from the starting population information. Given the new parameters, the total population sample size can now be calculated automatically and becomes a model-generated parameter.

```
void PreSimulation() { // Weight ActorWeight = StartPopSize / StartPopSampleSize;
```

```

// Total Immigrant population
ImmiPopSize = 0.0;
if (ModelImmigration) // If immigration is switched on
{
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
        {
            ImmiPopSize = ImmiPopSize + NumberImmigrants[nYear][nSex];
            NumberImmigrantsTable[nYear][nSex] = NumberImmigrants[nYear][nSex];
        }
    }
}

//Total population
TotalPopSize = ImmiPopSize + StartPopSize;
TotalPopSampleSize = (long)(TotalPopSize / ActorWeight);

ImmiPopSampleSize = TotalPopSampleSize - StartPopSampleSize;

};

```

7.11.4. Discussion: Model type conversions

This discussion contains the code of alternative model versions.

At any step 0-8, the model can be converted from case based to time based by replacing the model engine (the code in DYNAMIS-POP-MRT.mpp).

At any step 1-9, the model can be converted to a model reading in a starting population file instead of sampling starting characteristics from a distributional table. This is done by replacing the starting population module.

Note that some settings in ModelSettings.mpp have to be modified when changing the model type. The model settings file changes by modeling step and its alternative versions are documented at each step.

The DYNAMIS-POP-MRT.mpp file for time-based models (all steps)

The full code of DYNAMIS-POP-MRT.mpp for time-based models is detailed below. Note that this code will not be changed in the following modeling steps. Model developers do not need to modify or understand this code.

```

//LABEL(DYNAMIS-POP-MRT, EN) Simulation Engine for a time-based model

/* NOTE(DYNAMIS-POP-MRT, EN)

    This module is part of the time-based Template at step 0

    This module is the simulation engine for a time-based model.
    It contains core simulation functions and definitions.
    When starting from a template (or using the Modgen wizard), model developers
    typically do
    not have to modify (or understand) the code of this file.

    The file contains the definition of the model type: in our case a time-based
    continuous time model
    The second part consists of the functions handling the simulation

    Case-based versions of DYNAMIS-POP-MRT can be changed into a time-based model by

```

```

exchanging this file resp.
  by copy-pasting the code of this file into the DYNAMIS-POP-MRT.mpp file replacing
  the code.

  The code contains one parameter - StartingPopulationSize - which is necessary in
  time-based models
  (as it and not part of scenario-settings as in case-based models).
  Note that this parameter has to be added to the .dat file when changing a
  case-based model into a time-based one
  by adding: int StartingPopulationSize = 1000;
*/

// Definition of basic model characteristics

version 1, 0, 0, 0;           // The model version number
model_type time_based, just_in_time; // The model type
time_type double;           // The data type used to represent time
options packing_level = 2;   // An option reducing memory use at the
expense of speed

// Other data types

real_type      float;
counter_type   ushort;
integer_type   short;
index_type     ulong;

// Supported languages

languages
{
  EN // English
};

// Model strings

string S_MODEL_FINISH;      //EN Finish
string S_MODEL_REPLICATE;   //EN Replicate
string S_MODEL_SIMULATION;  //EN Simulation
string S_MODEL_START;       //EN Start

// Parameters

parameters
{
  model_generated logical model_is_case_based; //EN Model is case based
};

void PreSimulation()
{
  model_is_case_based = FALSE;
};

// The Simulation function is called by Modgen to simulate a replicate.

void Simulation()
{
  // Buffer for reporting progress
  const size_t nBufSize = 255;

```

```

TCHAR szBuffer[nBufSize];

// Note replicate number for progress reporting
int nReplicate = GetReplicate();

// Create the starting population
_stprintf_s(szBuffer, nBufSize, _T("%s %d: %s"),
ModelString("S_MODEL_REPLICATE"), nReplicate, ModelString("S_MODEL_START"));
ProgressMessage( szBuffer );

for ( long nJ = 0; nJ < TotalPopSampleSize; nJ++ )
{
    Person *paPerson = new Person();
    paPerson->Start(nJ, NULL );
}

_stprintf_s(szBuffer, nBufSize, _T("%s %d: %s"),
ModelString("S_MODEL_REPLICATE"), nReplicate, ModelString("S_MODEL_SIMULATION"));
ProgressMessage( szBuffer );

// event loop
double dCurrentTime = TIME_INFINITE;
double dStartTime = TIME_INFINITE;

int nLastProgressPercent = -1;
int nThisProgressPercent = -1;

while ( !gpoEventQueue->Empty() )
{
    // get the time of next event, verify against the simulation end
    dCurrentTime = gpoEventQueue->NextEvent();

    // Note the start time (time of first event) for progress indicator
    if ( dStartTime == TIME_INFINITE )
    {
        dStartTime = dCurrentTime;
    }

    if ( dCurrentTime > SIMULATION_END() || gbInterrupted || gbCancelled ||
gbErrors )
    {
        if (dCurrentTime > SIMULATION_END())
        {
            // age all actors to the simulation end time
            gpoEventQueue->WaitUntilAllActors( SIMULATION_END() );
        }

        _stprintf_s(szBuffer, nBufSize, _T("%s %d: %s"),
ModelString("S_MODEL_REPLICATE"), nReplicate, ModelString("S_MODEL_FINISH"));
        ProgressMessage( szBuffer );

        gpoEventQueue->FinishAllActors();
    }
    else
    {
        // age all actors to the time of next event
        gpoEventQueue->WaitUntil( dCurrentTime );
    }
}

```

```

        // implement the next event
        gpoEventQueue->Implement();
    }

    // Update progress indicator only if the integer percentage complete changes
    // (updates to the progress bar at every event are expensive).
    nThisProgressPercent = (int)( 100 * ( dCurrentTime - dStartTime ) /
        ( SIMULATION_END() - dStartTime ) );

    if ( nThisProgressPercent > nLastProgressPercent )
    {
        TimeReport( dCurrentTime ); // update simulation progress
        nLastProgressPercent = nThisProgressPercent;
    }
}

```

The StartPopFile.mpp module for reading in a starting file

The file StartPopFile.mpp replaces the file StartPopSampling.mpp (which has to be removed). The code of this file is discussed at Step 9 when preparing the base model for model extensions requiring a time-based model with a starting population file.

```

//LABEL(StartPopFile, EN) Starting population File

/* NOTE(StartPopFile, EN)

Added at step 9 replacing StartPopSampling.mpp

This replacement can be made at any previous modeling step

This file replaces StartPopSampling.mpp and converts the model from a model that
samples
starting characteristics to a model that reads in a micro-population file.
Variables are time of birth, sex, and region

*/

input_csv in_csv; //EN Microdata input csv
object

classification PERSON_MICRODATA_COLUMNS //EN fields in the
microdata input file
{
    PMC_BIRTH, //EN Time of birth
    PMC_SEX, //EN Sex
    PMC_PROVINCE //EN Province
};

parameters
{
    file MicroDataInputFile; //EN File name of
starting population
};

parameter_group PG00_MicroDataInputFile //EN Starting Population
file
{
    MicroDataInputFile

```

```

};

actor Person
{
    void GetStartCharacteristics(long i_lLine);           //EN Read characteristics
    at birth
};

void Person::GetStartCharacteristics(long i_lLine)
{
    // Read record
    in_csv.read_record(i_lLine);

    // Assigns values from record
    time_of_birth = in_csv[PMC_BIRTH];                 // time of
    birth
    if ((int)in_csv[PMC_SEX] == 1) sex = MALE; else sex = FEMALE; // sex
    province_nat = (PROVINCE_NAT)(int)in_csv[PMC_PROVINCE]; // province
}

void PreSimulation()
{
    in_csv.open(MicroDataInputFile);
}

void PostSimulation()
{
    in_csv.close();
}

```

7.12. Step 12: Sampling from a detailed starting population

7.12.1. Overview

In this step, we replace the starting population file with its final version. The file has more variables and also includes a weight variable. File size and sample size do not have to correspond; if the file size is larger than the selected sample size, records are sampled. If the sample size exceeds the file size, observations are cloned.

7.12.2. Concepts

In this step, we add a new actor type, “Observation.” Observations correspond with (sampled) file records and have integer weights. Some of the added functionality of time-based models is introduced.

Adding a new actor type

Models can have any number of actor types. New actors are introduced simply by an actor statement block, containing actor states and at least the two functions, Start() and Finish(). For example, the minimum code for adding an actor Observation is:

```

actor Observation                                     //EN Actor Observations
{
    void Start();                                     //EN Starts the actor
    void Finish();                                    //EN Destroys the actor
};

```

```
// Implementation
void Observation::Start()
{
    // code can be added here
}

void Observation::Finish()
{
    // code can be added here
}
```

New actors can be created by other actors in events (e.g., births are created by mothers) or can be created by the simulation engine at the start of the simulation.

Code example from a Simulation() function, reading in a file, and creating an actor for each record, thereby passing the current record object as a parameter:

```
// Create observations
for (long nJ = 0; nJ < MicroDataInputFileSize; nJ++)
{
    in_csv.read_record(nJ);
    Observation *paObservation = new Observation();
    paObservation->Start(in_csv);
}
```

Actor Sets

One of the most powerful concepts available in time-based modes are record sets, a collection of actors of one type (e.g., persons) that is maintained automatically. Record sets can have dimensions, e.g., age or province of residence. At any change of individual characteristics (e.g., birthdays, migration), the change in group membership is updated automatically. Actor sets can contain filters, making membership dependent on individual characteristics. When characteristics change, the membership is updated automatically. Actor sets can be ordered.

```
//EN All Observation actors
actor_set Observation asAllObservations;

//EN Persons by age and sex
actor_set Person asPersonsByAgeAndSex[integer_age][sex]

//EN 8 year old persons by education of mother
actor_set Person asPersonsAge08[educ_mother] filter integer_age == 8;

//EN Persons eligible for marriage by sex, ordered by income
actor_set Person asEligibleOrdered[sex] filter wants_to_marry order income;
```

Actor sets are very useful to find out how many actors (with specific characteristics) exist, and to find a specific (e.g. the first) or a random actor of the set.

```
// How many observations are available?
int nNumberObservations = asAllObservations->Count();

// Create a pointer to a random observation:
Observation * prRandomObservation = asAllObservations->GetRandom(RandUniform(2));

// Create a pointer to the first observation:
Observation * prFirstObservation = asAllObservations->Item(0);
```

```
// Use the pointer to access a state of the actor
time_of_birth = prRandomObservation->obs_birth;

// Kill the first actor of the set
prFirstObservation->Finish();
```

7.12.3. How to reproduce this modeling step

File modifications and extensions are made in three files, most importantly in StartPopFile.mpp, where the new actor Observation is implemented and used. The simulation engine DYNAMIS-POP-MRT.mpp is changed to create observation actors from the input file. We also change one line in the Start() function in PersonCore.mpp.

Changes to file operations for allowing multi-threading

So far, all file input operations (i.e., creating the file object, opening the input file, reading records, and closing the file) were contained in the StartPopFile.mpp module. To allow multi-threading, or running multiple replicas of the same model simultaneously, we move this code into the Simulation() function, which automatically handles these issues.

```
void Simulation()
{
    // Create and open the input data file
    input_csv in_csv;                               // Microdata input csv object
    in_csv.open(MicroDataInputFile);               // Open the microdata input file

    [...]

    // Close teh microdata input file
    in_csv.close();
}
```

Input file variables and file lenght (StartPopFile.mpp)

We extend the classification to contain the full set of variables of the model.

```
classification PERSON_MICRODATA_COLUMNS           //EN fields in the microdata input
file
{
    PMC_WEIGHT,                                   //EN Weight
    PMC_BIRTH,                                   //EN Time of birth
    PMC_SEX,                                     //EN Sex
    PMC_PROVINCE,                                //EN Province
    PMC_EDUC,                                    //EN Education
    PMC_POB,                                     //EN Province of birth
    PMC_UNION,                                   //EN Union formation time
    PMC_PARITY,                                  //EN Number of births
    PMC_LASTBIR                                 //EN Time of last birth
};
```

Also, as we allow for weighting and do not require correspondence between sample size and input file length, we add two parameters:

```
parameters
{
    file          MicroDataInputFile;           //EN File name of starting population
    long          MicroDataInputFileSize;       //EN File size of starting population
    logical       UseWeightsInFile;            //EN Use weights in starting
population file
```

```

};

parameter_group PG00_MicroDataInputFile          //EN Starting Population file
{
    MicroDataInputFile, MicroDataInputFileSize,
    UseWeightsInFile
};

```

Accordingly, this code has to be removed from the StartPopFile.mpp module, and the whole PreSimulation() and PostSimulation() functions become obsolete and can be removed.

The new actor type Observation (StartPopFile.mpp)

“Observation” can be created just by an actor code block of its name, containing its states and functions. States correspond to the fields of the input file. The Start() function takes a record from the file as input parameter. Additionally, we create a state for the integer weight, which will contain information as to how many Person actors corresponding with an observation should be created. The Function WeightOrKill() determines the integer weight and removes the observation if the weight is 0, i.e., the observation is not needed.

```

actor Observation                                //EN Actor Observations
{
    // values from file
    double   pmc[PERSON_MICRODATA_COLUMNS];    //EN Person micro record columns

    // states
    integer  int_weight = { 1.0 };             //EN Integer Weight

    // functions
    void Start(const input_csv& input);        //EN Starts the actor
    void Finish();                             //EN Destroys the actor
    void WeightOrKill();                       //EN Decides if an observation stays
    in sample and how often
};

```

The Start() function simply initializes the states of the Observation; the implementation of Finish() is empty.

```

void Observation::Start(const input_csv& in_csv)
{
    for (int nJ = 0; nJ < SIZE(PERSON_MICRODATA_COLUMNS); nJ++)
    {
        pmc[nJ] = in_csv[nJ];
    }
};

void Observation::Finish(){};

```

The function WeightOrKill() calculates integer weights for each record in the file. Based on the non-integer part of the weight, a random number is used to decide if weights are rounded up or down. Actors with a resulting integer weight 0 are removed. Weights add up to the sample size parameter.

```

void Observation::WeightOrKill()
{
    double  dWeight = 0.0;
    int     nWeight = 0;

    if (!UseWeightsInFile) dWeight = (double)StartPopSampleSize /
(double)MicroDataInputFileSize;
    else dWeight = pmc[PMC_WEIGHT] * (double)StartPopSampleSize / StartPopSize;
};

```

```

    nWeight = (int)dWeight;
    dWeight = dWeight - nWeight;
    if (RandUniform(3) < dWeight) nWeight++;
    int_weight = nWeight;
    if (int_weight == 0) Finish();
}

```

To be able to sample from the available observations, we create an actor set of all observations.

```

actor_set Observation asAllObservations;           //EN All Observations

```

Using the new Observation actor for sampling starting characteristics (StartPopFile.mpp)

As the final change in the module StartPopFile.mpp, we modify the existing **GetStartCharacteristics()** function, which no longer takes a parameter. Instead of reading a record with a given number, we now take an observation from the actor set asAllObservations.

The starting characteristics of the Person actor are now initialized using the selected Observation actor.

Once the Observation is “used,” its integer weight is reduced by one, or, if the weight is one, the observation is removed automatically from the Actor Set by calling its Finish() function. Note that as the weighted number of records corresponds with the sample size, we could have taken the first Observation actor of the set instead of sampling.

```

actor Person
{
    void GetStartCharacteristics();           //EN Get characteristics
at birth
};

void Person::GetStartCharacteristics()
{
    // Sample an observation
    Observation * prObservation = asAllObservations->GetRandom(RandUniform(2));

    // Assigns values from observation
    time_of_birth = prObservation->pmc[PMC_BIRTH];
    if ((int)prObservation->pmc[PMC_SEX] == 1) sex = MALE; else sex = FEMALE;
    province_nat = (PROVINCE_NAT) (int)prObservation->pmc[PMC_PROVINCE];

    // Decrement the integer weight of the observation actor and kill it if not
    // needed anymore
    prObservation->int_weight--;
    if (prObservation->int_weight <= 0 ) prObservation->Finish();
}

```

Changes in the Simulation() function (DYNAMIS-POP-MRT.mpp)

After opening the input file, and before creating the Person actors, we create the Observation actors by looping through the micro-data input file. After creating an observation, we decide if and how often it is to be used by setting an integer weight. This is done by calling WeightOrKill(), which also removes an actor and releases its memory space immediately if it is not used.

```

// Create observations
for (long nJ = 0; nJ < MicroDataInputFileSize; nJ++)
{
    in_csv.read_record(nJ);
    Observation *paObservation = new Observation();
}

```

```

paObservation->Start(in_csv);
paObservation->WeightOrKill();
}

```

Changes in the Start() function

Because a random element is contained in the decision if and how often each observation in the starting population file is used, the resulting starting population size based on integer-weighted observations becomes a random variable that is very close, but not necessarily equal to, the StartPopSampleSize Parameter. To decide if an actor stems from the starting population, we replace the condition “PersonNr < StartPopSampleSize” with “asAllObservations->Count() > 1” to ensure we did not run out of observations.

```

else if (asAllObservations->Count() > 1) // Person from start population
{
    person_type = PT_START;
}

```

Note that the PersonNr as a parameter of the Start() function becomes obsolete and can be removed. Also, as an actor set is used, it is not possible to convert back to a case-based model and the code could be cleaned up for removing case-based specific code.

7.13. Step 13: Primary Education

7.13.1. Overview

This module implements primary education. Users can specify entry and graduation ages, as well as probabilities to enter and to graduate by year of birth, sex, and province of birth. Optionally, proportional factors can be specified accounting for mother’s education. If these factors are used, the outcome calibrates itself for a selected year to the initial outcome.

The primary education system is implemented as a “PrimarySchool” actor, who at the change of each school year selects potential students for entering school for graduation.

7.13.2. Concepts

This step provides a rich set of examples as to the use of actor sets and the communication between two types of actors: individual persons and an actor representing the primary school system.

- **Efficiency:** Avoiding large numbers of individual level events happening at the same time.
- **Data imputation:** Sampling characteristics from a donor population.
- **Alignment** of aggregated results respecting relative differences in individual probabilities.

Gaining efficiency by avoiding large numbers of individual level events happening at the same time

In this step, we add a new actor type, “PrimarySchool.” This actor is predominantly introduced for efficiency gains. Instead of each person individually scheduling school entry and graduation events all happening at the same time each year, one single central actor calls a school year change event. The PrimarySchool actor loops through the population of all people eligible for school entry and graduation and calls functions on the individual level to decide school transitions. The eligible population is automatically maintained by actor sets.

The following is a simplified example of how school entry can be modeled in such a way:

```

//EN Primary education entry age cohort

```

```

actor_set Person asPrimaryEntryCohort filter integer_age == AgeEnterPrimary;

// A function to be called at each school year change
void PrimarySchool::EnterPrimary()
{
    // loop through all eligible actors
    for (long nJ = 0; nJ < asPrimaryEntryCohort->Count(); nJ++)
    {
        Person * prPerson = new Person();
        prPerson = asAllPersons->Item(nJ);

        prPerson->AnIndividualLevelFunctionToDecideIndividualSchoolEntry();
    }
}

```

Sampling characteristics from a donor population

Actor sets are a very powerful and straightforward mechanism for sampling characteristics from a donor population, for example, for imputing missing information by sampling from a population of people with similar characteristics. An example from this modeling step is the imputation of education of new immigrants if, at the moment of immigration, they are beyond graduation age and born before the year of birth range for which education is modeled in the application.

```

[...]

// (c) sample from other foreign born of same age, sex and province of residence
// if no foreignborn with same characteristics available, sample from all
// residents of same age and sex if no donor is found, assign education randomly
else
{
    Person * prPerson = new Person;
    int nPopSize = asForeignersAgeSexProv[integer_age][sex][province_nat]
        ->Count();
    if (nPopSize > 0)
    {
        prPerson = asForeignersAgeSexProv[integer_age][sex][province_nat]
            ->GetRandom(RandUniform(19));
    }
    else
    {
        nPopSize = asResidentsAgeSex[integer_age][sex]->Count();
        if (nPopSize > 0) prPerson = asResidentsAgeSex[integer_age][sex]
            ->GetRandom(RandUniform(20));
    }
    // a donor was found: assign donor's primary entry level
    if (nPopSize > 0 && prPerson->primary_level != PL_NO) primary_level = PL_ENTER;
    // no donor was found; depending on samlpe size this should never or rarely
    happen
    else if (RandUniform(21) < 0.5) primary_level = PL_ENTER;
}

```

Aligning model results

One of the strengths of time-based models is the ability to align aggregated output to external targets, while respecting relative differences in probabilities or risks. This is used at this step to allow the introduction of additional relative factors—log odds for the influence of mother’s education—without changing the aggregated output for the year of introduction. By aligning results this way, a calibration factor is derived and applied for the following years.

The following illustration is the `PrimarySchoolEntry()` function introduced at this modeling step. This function is called by the Primary School Actor every year to select children to enroll in primary school. The user can choose two model variants, one with and one without accounting for mother's education. In the latter case, the outcome is aligned to the initial outcome for the first year in which mother's education is introduced:

```
void PrimarySchool::PrimarySchoolEntry()
{
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        for (int nProv = 0; nProv < SIZE(PROVINCE_INT); nProv++)
        {
            // Size of relevant sub-populations
            int nMotherEd0 = asPrimaryEntryCohort[nSex][nProv][PL_NO]->Count();
            int nMotherEd1 = asPrimaryEntryCohort[nSex][nProv][PL_ENTER]->Count();
            int nMotherEd2 = asPrimaryEntryCohort[nSex][nProv][PL_GRAD]->Count();
            int nPerson = nMotherEd0 + nMotherEd1 + nMotherEd2;

            // Calculate probabilities with all adjustments
            if (nPerson > 0)
            {
                double dCalibrationFactor = 0.0;
                double dMotherFactorEduc0 = 0.0;
                double dMotherFactorEduc1 = 0.0;
                double dMotherFactorEduc2 = 0.0;
                double dCenter;

                // overall probability if not accounting for mother seducation
                double dProb =
                    StartPrimary[nSex][RANGE_POS(YOB_START_PRIMARY, school_year -
AgeEnterPrimary)][nProv];

                // account for mothers education and calibrate results in year in
which introduced
                if (UseMothersEducation && school_year >= CalibratedYear +
AgeEnterPrimary)
                {
                    dMotherFactorEduc0 = log(OddsMotherEnterPrimary[nSex][PL_NO]);
                    dMotherFactorEduc1 = log(OddsMotherEnterPrimary[nSex][PL_ENTER]);
                    dMotherFactorEduc2 = log(OddsMotherEnterPrimary[nSex][PL_GRAD]);

                    // calibration year
                    if (school_year == CalibratedYear + AgeEnterPrimary)
                    {
                        double dWeight0 = (double)nMotherEd0 / (double)nPerson;
                        double dWeight1 = (double)nMotherEd1 / (double)nPerson;
                        double dWeight2 = (double)nMotherEd2 / (double)nPerson;

                        int nIterations = 0;
                        double dResult = 500.0;
                        double dTarget = dProb;
                        double dLower = -10.0;
                        double dUpper = 10.0;
                        while (abs(dResult - dTarget) > 0.0001 && nIterations <
10000)
                        {
                            nIterations++;
                            dCenter = (dLower + dUpper) / 2.0;

```

```

        dResult = dWeight0 * AdjustedProbability(dProb,
dMotherFactorEduc0, dCenter)
        + dWeight1 * AdjustedProbability(dProb,
dMotherFactorEduc1, dCenter)
        + dWeight2 * AdjustedProbability(dProb,
dMotherFactorEduc2, dCenter);

        if (dTarget > dResult) dLower = dCenter;
        else dUpper = dCenter;
    }
    dCalibrationFactor = dCenter;

    LastProbabilityPrimaryEntry[nSex][nProv] = dProb;
    CalibrationFactorPrimaryEntry[nSex][nProv] =
dCalibrationFactor;
    }
    // after calibrated year
    else
    {
        dCalibrationFactor =
CalibrationFactorPrimaryEntry[nSex][nProv];
        dProb = LastProbabilityPrimaryEntry[nSex][nProv];
    }
    }
    // decide for each person in entry cohort, if entering school
    for (int nJ = 0; nJ < nMotherEd0; nJ++)
asPrimaryEntryCohort[nSex][nProv][PL_NO]->Item(nJ)->
    DecidePrimaryEntry(AdjustedProbability(dProb, dMotherFactorEduc0,
dCalibrationFactor));
    for (int nJ = 0; nJ < nMotherEd1; nJ++)
asPrimaryEntryCohort[nSex][nProv][PL_ENTER]->Item(nJ)->
    DecidePrimaryEntry(AdjustedProbability(dProb, dMotherFactorEduc1,
dCalibrationFactor));
    for (int nJ = 0; nJ < nMotherEd2; nJ++)
asPrimaryEntryCohort[nSex][nProv][PL_GRAD]->Item(nJ)->
    DecidePrimaryEntry(AdjustedProbability(dProb, dMotherFactorEduc2,
dCalibrationFactor));
    }
    }
}

```

7.13.3. How to reproduce this modeling step

At this step, we add a module PrimaryEducation.mpp. Changes in other modules are very minor.

The PrimaryEducation.mpp module

Actor set and type definitions

```

//EN Primary education entry age cohort
actor_set Person asPrimaryEntryCohort[sex][province_birth][educ_mother]
    filter set_alive && integer_age == AgeEnterPrimary;

//EN Primary education graduation age cohort
actor_set Person asPrimaryGraduationCohort[sex][province_birth][educ_mother]
    filter set_alive && integer_age == AgeGradPrimary && primary_level != PL_NO;

//EN All Persons

```

```

actor_set Person asAllPersons filter set_alive;

//EN Foreign born by age, sex and province
actor_set Person asForeignersAgeSexProv[integer_age][sex][province_nat]
filter set_alive && province_birth == PI_PROV13;

//EN Residents by age, sex and province
actor_set Person asResidentsAgeSex[integer_age][sex] filter set_alive;

//EN Foreign born who at least entered primary education by age, sex and province
actor_set Person asForeignersPrimAgeSexProv[integer_age][sex][province_nat]
filter set_alive && province_birth == PI_PROV13 && primary_level != PL_NO;

// EN Residents who at least entered primary education
actor_set Person asResidentsPrimAgeSex[integer_age][sex] filter set_alive &&
primary_level != PL_NO;

classification PRIMARY_LEVEL //EN Primary
Education level
{
    PL_NO, //EN Never entered
    primary school
        PL_ENTER, //EN Entered primary
        school
            PL_GRAD //EN Graduated from
primary school
};

range YOB_START_PRIMARY { 2001, 2063 }; //EN Year of birth
range YOB_GRAD_PRIMARY { 1997, 2063 }; //EN Year of birth

```

Model parameters

The primary school system is parameterized by a school entry age, a graduation age, and the time in the calendar year, a new school year starts. School entry and graduation is decided according to probabilities by year of birth, province of birth, and sex. Additionally, odds ratios by mothers' education can be specified. Their use is optional, when used, results are aligned to the results without mothers' education for a selected year at which this additional variable is introduced.

```

parameters
{
    int      AgeEnterPrimary; //EN School entry
age primary
    int      AgeGradPrimary; //EN Graduation age
    primary
        double StartOfSchoolYear; //EN Start at
school year (e.g. 0.5 = July)
        double StartPrimary[SEX][YOB_START_PRIMARY][PROVINCE_INT]; //EN Probability to
start primary school
        double GradPrimary[SEX][YOB_GRAD_PRIMARY][PROVINCE_INT]; //EN Probability to
graduate from primary school
        double OddsMotherEnterPrimary[SEX][PRIMARY_LEVEL]; //EN Odds by
mother's education: primary
        double OddsMotherGradPrimary[SEX][PRIMARY_LEVEL]; //EN Odds by
mother's education: secondary
    int      CalibratedYear; //EN Calibrated year
introducing mothers education
        logical UseMothersEducation; //EN Use mother's

```

```
education
};
```

Personal level states and functions

The key state modeled in this module is the primary school level. There are functions to decide school entry and graduation for given probabilities and a function to initialize education of immigrants at the moment they enter the country.

```
actor Person
{
    PRIMARY_LEVEL    primary_level = { PL_NO };           //EN Primary school
status
    PRIMARY_LEVEL    educ_mother = { PL_NO };           //EN Mothers primary
school status

    void DecidePrimaryEntry(double dProb);              //EN Decides & sets
primary school entry
    void DecidePrimaryGrad(double dProb);              //EN Decides & sets
primary graduation
    void SetImmigrantPrimaryEducationStatus();         //EN Assign primary
education at immigration
    hook SetImmigrantPrimaryEducationStatus, SetAliveEvent;
};
```

The actor Primary School definitions and basics: Start, Finish, School year change

As all actors, the primary school actor is declared in an actor code block containing its states and functions. Mandatory functions are Start() and Finish(). The actor is created at the start of the simulation. The actor has one single event—the change of school year—in which two key functions of this module are called PrimarySchoolEntry() and PrimarySchoolGraduation().

```
actor PrimarySchool                                     //EN Primary School
Actor
{
    int      school_year = { 0 };                       //EN Current school
year
    TIME     next_school_year = { TIME_INFINITE };      //EN Next school year
    TIME     assign_primary_time = { TIME_INFINITE };   //EN Initial
assignment of primary status

    double   CalibrationFactorPrimaryEntry[SEX][PROVINCE_INT]; //EN Calibration factor
primary entry
    double   LastProbabilityPrimaryEntry[SEX][PROVINCE_INT]; //EN Last primary entry
probability
    double   CalibrationFactorPrimaryGrad[SEX][PROVINCE_INT]; //EN Calibration factor
primary graduation
    double   LastProbabilityPrimaryGrad[SEX][PROVINCE_INT]; //EN Last primary entry
probability

    event    timeNextSchoolYearEvent, NextSchoolYearEvent; //EN Next school year
event
    event    timeAssignInitialPrimary, AssignInitialPrimary; //EN Assigns initial
school level at start

    void     PrimarySchoolEntry();                     //EN Decides who
enters primary education
    void     PrimarySchoolGraduation();               //EN Decides who
graduates from primary education
```

```

//EN Calculate adjusted probability adding relative factors
double AdjustedProbability(double dProb, double dLogOddEduc, double
dLogOddAdjust);

void Start(); //EN Start the
primary school actor
void Finish(); //EN Destroy the
primary school actor
};

void PrimarySchool::Start()
{
    time = MIN(SIM_YEAR_RANGE); // Create at start of
simulation
    school_year = MIN(SIM_YEAR_RANGE) - 1; // Still 'old' school year
    next_school_year = WAIT(0.5); // Next school year starts
mid year
    assign_primary_time = WAIT(0.0); // Assign initial proiary
status in population
};

void PrimarySchool::Finish(){}

TIME PrimarySchool::timeNextSchoolYearEvent()
{
    return next_school_year;
}

/* NOTE(PrimarySchool.NextSchoolYearEvent, EN)
The primary school actor has just this one event, the change of school year.
At each school year change, a selection is made which potential students enter
primary education, and which students graduate.
*/
void PrimarySchool::NextSchoolYearEvent()
{
    school_year++; // update school year
    PrimarySchoolEntry(); // decide who enters primary
school
    PrimarySchoolGraduation(); // decide who graduates from
primary school
    next_school_year = WAIT(1.0); // set clock for next school year
}

```

Initialization of the primary education status in the starting population

This event assigns the initial primary education status at the beginning of the simulated time period. For given parameters of school entry and graduation age, values in the file are ignored and reset (if the person is too young for having entered/graduated from primary school), reassigned by models (if the recorded outcome of schooling is not necessarily the final outcome), or kept.

```

void PrimarySchool::AssignInitialPrimary()
{
    // loop through all actors
    for (long nJ = 0; nJ < asAllPersons->Count(); nJ++)
    {
        Person * prPerson = new Person();
        prPerson = asAllPersons->Item(nJ);

        // Entry to primary education
    }
}

```

```

// (a) persons below school entry age reset education on file to non
if (prPerson->age < AgeEnterPrimary + StartOfSchoolYear)
{
    prPerson->primary_level = PL_NO;
}
// (b) primary education: modeled if yob in parameterized range
else if (prPerson->year_of_birth >= MIN(YOB_START_PRIMARY))
{
    prPerson->primary_level = PL_NO; // reset
    if (RandUniform(8) < StartPrimary[prPerson->sex]
        [RANGE_POS (YOB_START_PRIMARY,
prPerson->year_of_birth)][prPerson->province_birth])
        prPerson->primary_level = PL_ENTER;
}
// (c) old enough: keep primary education from file (do nothing)
else {}

// Graduate from primary education
// (a) persons below school graduation age reset graduations on record
if (prPerson->age < AgeGradPrimary + StartOfSchoolYear &&
prPerson->primary_level == PL_GRAD)
{
    prPerson->primary_level = PL_ENTER;
}
// (b) primary education graduation: modeled if yob in parameterized range
else if (prPerson->age >= AgeGradPrimary + StartOfSchoolYear
    && prPerson->year_of_birth >= MIN(YOB_GRAD_PRIMARY) &&
prPerson->primary_level != PL_NO)
{
    prPerson->primary_level = PL_ENTER; // reset
    if (RandUniform(15) < GradPrimary[prPerson->sex]
        [RANGE_POS (YOB_GRAD_PRIMARY,
prPerson->year_of_birth)][prPerson->province_birth])
        prPerson->primary_level = PL_GRAD;
}
// (c) old enough: keep primary education from file (do nothing)
else {}
}
assign_primary_time = TIME_INFINITE; //do just once
}

```

Primary school entry

This function is called by the Primary School Actor every year to select children to enroll in primary school. The user can choose two model variants, one with and one without accounting for mother's education. In the latter case, the outcome is aligned to the initial outcome for the first year in which mother's education is introduced.

```

void PrimarySchool::PrimarySchoolEntry()
{
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        for (int nProv = 0; nProv < SIZE(PROVINCE_INT); nProv++)
        {
            // Size of relevant sub-populations
            int nMotherEd0 = asPrimaryEntryCohort[nSex][nProv][PL_NO]->Count();
            int nMotherEd1 = asPrimaryEntryCohort[nSex][nProv][PL_ENTER]->Count();
            int nMotherEd2 = asPrimaryEntryCohort[nSex][nProv][PL_GRAD]->Count();
            int nPerson = nMotherEd0 + nMotherEd1 + nMotherEd2;

```

```

// Calculate probabilities with all adjustments
if (nPerson > 0)
{
    double dCalibrationFactor = 0.0;
    double dMotherFactorEduc0 = 0.0;
    double dMotherFactorEduc1 = 0.0;
    double dMotherFactorEduc2 = 0.0;
    double dCenter;

    // overall probability if not accounting for mother seducation
    double dProb =
        StartPrimary[nSex][RANGE_POS(YOB_START_PRIMARY, school_year -
AgeEnterPrimary)][nProv];

    // account for mothers education and calibrate results in year in
which introduced
    if (UseMothersEducation && school_year >= CalibratedYear +
AgeEnterPrimary)
    {
        dMotherFactorEduc0 = log(OddsMotherEnterPrimary[nSex][PL_NO]);
        dMotherFactorEduc1 = log(OddsMotherEnterPrimary[nSex][PL_ENTER]);
        dMotherFactorEduc2 = log(OddsMotherEnterPrimary[nSex][PL_GRAD]);

        // calibration year
        if (school_year == CalibratedYear + AgeEnterPrimary)
        {
            double dWeight0 = (double)nMotherEd0 / (double)nPerson;
            double dWeight1 = (double)nMotherEd1 / (double)nPerson;
            double dWeight2 = (double)nMotherEd2 / (double)nPerson;

            int nIterations = 0;
            double dResult = 500.0;
            double dTarget = dProb;
            double dLower = -10.0;
            double dUpper = 10.0;
            while (abs(dResult - dTarget) > 0.0001 && nIterations <
10000)
            {
                nIterations++;
                dCenter = (dLower + dUpper) / 2.0;

                dResult = dWeight0 * AdjustedProbability(dProb,
dMotherFactorEduc0, dCenter)
                    + dWeight1 * AdjustedProbability(dProb,
dMotherFactorEduc1, dCenter)
                    + dWeight2 * AdjustedProbability(dProb,
dMotherFactorEduc2, dCenter);

                if (dTarget > dResult) dLower = dCenter;
                else dUpper = dCenter;
            }
            dCalibrationFactor = dCenter;

            LastProbabilityPrimaryEntry[nSex][nProv] = dProb;
            CalibrationFactorPrimaryEntry[nSex][nProv] =
dCalibrationFactor;
        }
        // after calibrated year

```



```

        if (UseMothersEducation && school_year >= CalibratedYear +
AgeGradPrimary)
    {
        dMotherFactorEduc0 = log(OddsMotherGradPrimary[nSex][PL_NO]);
        dMotherFactorEduc1 = log(OddsMotherGradPrimary[nSex][PL_ENTER]);
        dMotherFactorEduc2 = log(OddsMotherGradPrimary[nSex][PL_GRAD]);

        // calibration year
        if (school_year == CalibratedYear + AgeGradPrimary)
        {
            double dWeight0 = (double)nMotherEd0 / (double)nPerson;
            double dWeight1 = (double)nMotherEd1 / (double)nPerson;
            double dWeight2 = (double)nMotherEd2 / (double)nPerson;

            int nIterations = 0;
            double dResult = 500.0;
            double dTarget = dProb;
            double dLower = -10.0;
            double dUpper = 10.0;
            while (abs(dResult - dTarget) > 0.0001 && nIterations <
10000)
            {
                nIterations++;
                dCenter = (dLower + dUpper) / 2.0;

                dResult = dWeight0 * AdjustedProbability(dProb,
dMotherFactorEduc0, dCenter)
                    + dWeight1 * AdjustedProbability(dProb,
dMotherFactorEduc1, dCenter)
                    + dWeight2 * AdjustedProbability(dProb,
dMotherFactorEduc2, dCenter);

                if (dTarget > dResult) dLower = dCenter;
                else dUpper = dCenter;
            }
            dCalibrationFactor = dCenter;

            LastProbabilityPrimaryGrad[nSex][nProv] = dProb;
            CalibrationFactorPrimaryGrad[nSex][nProv] =
dCalibrationFactor;
        }
        // after calibrated year
        else
        {
            dCalibrationFactor =
CalibrationFactorPrimaryGrad[nSex][nProv];
            dProb = LastProbabilityPrimaryGrad[nSex][nProv];
        }
        // decide for each primary student if graduating

        for (int nJ = 0; nJ < nMotherEd0; nJ++)
asPrimaryGraduationCohort[nSex][nProv][PL_NO]->
            Item(nJ)->DecidePrimaryGrad(AdjustedProbability(dProb,
dMotherFactorEduc0, dCalibrationFactor));
        for (int nJ = 0; nJ < nMotherEd1; nJ++)
asPrimaryGraduationCohort[nSex][nProv][PL_ENTER]->
            Item(nJ)->DecidePrimaryGrad(AdjustedProbability(dProb,
dMotherFactorEduc1, dCalibrationFactor));

```

```

        for (int nJ = 0; nJ < nMotherEd2; nJ++)
asPrimaryGraduationCohort[nSex][nProv][PL_GRAD]->
            Item(nJ)->DecidePrimaryGrad(AdjustedProbability(dProb,
dMotherFactorEduc2, dCalibrationFactor));
    }
}
}
}

```

Primary education status of immigrants

This function is called at the moment of immigration to assign a education status to the new immigrant. Depending on year of birth and age, there are three possibilities:

- The person's graduation age is younger than school entry .and currently has not entered primary.
- For the birth date, parameters are available; education status is modeled
- The person was born too far in the past and no parameters are available; characteristics are sampled from the current immigrant population of the same age, sex, and province of residence. If no donor is found, the sampling is extended to the national population of the same age and sex.

```

void Person::SetImmigrantPrimaryEducationStatus()
{
    if (person_type == PT_IMMIGRANT)
    {
        // ENTRY TO PRIMARY

        double dTimeToNextSchoolyear = StartOfSchoolYear - (time - calendar_year);
        if (dTimeToNextSchoolyear < 0) dTimeToNextSchoolyear = dTimeToNextSchoolyear
+ 1.0;

        // (a) younger than entry age at next school year change:
        if ((int)(age + dTimeToNextSchoolyear) <= AgeEnterPrimary) { /*do nothing*/ }

        // (b) parameters available for given year of birth
        else if (year_of_birth >= MIN(YOB_START_PRIMARY))
        {
            if (RandUniform(18) < StartPrimary[sex][RANGE_POS(YOB_START_PRIMARY,
year_of_birth)][province_birth])
                primary_level = PL_ENTER;
        }

        // (c) sample from other foreign born of same age, sex and province of
residence
        // if no foreignborn with same characteristics available, sample from all
residents of same age and sex
        else
        {
            Person * prPerson = new Person;
            int nPopSize =
asForeignersAgeSexProv[integer_age][sex][province_nat]->Count();
            if (nPopSize > 0) prPerson =
asForeignersAgeSexProv[integer_age][sex][province_nat]
->GetRandom(RandUniform(19));
            else
            {
                nPopSize = asResidentsAgeSex[integer_age][sex]->Count();
            }
        }
    }
}

```

```

        if (nPopSize > 0) prPerson = asResidentsAgeSex[integer_age][sex]
            ->GetRandom(RandUniform(20));
    }
    // a donor was found: assign donor's primary entry level
    if (nPopSize > 0 && prPerson->primary_level != PL_NO) primary_level =
PL_ENTER;
    // no donor was found; depending on samplpe size this should never or
rarely happen
    else if (RandUniform(21) < 0.5) primary_level = PL_ENTER;
    }

    // GRADUATION FROM PRIMARY

    if (primary_level == PL_ENTER)
    {
        double dTimeToNextSchoolyear = StartOfSchoolYear - (time -
calendar_year);
        if (dTimeToNextSchoolyear < 0) dTimeToNextSchoolyear =
dTimeToNextSchoolyear + 1.0;

        // (a) younger than graduation age at next school year change:
        if ((int)(age + dTimeToNextSchoolyear) <= AgeGradPrimary) { /* do nothing
*/ }

        // (b) parameters available for given year of birth
        else if (year_of_birth >= MIN(YOB_GRAD_PRIMARY))
        {
            if (RandUniform(22) < GradPrimary[sex][RANGE_POS(YOB_GRAD_PRIMARY,
year_of_birth)][province_birth])
                primary_level = PL_GRAD;
        }
        // (c) sample from other foreign born who entered primary of same age,
sex and province of residence
        // if no foreignborn with same characteristics available, sample from
all residents of same age and sex
        else
        {
            Person * prPerson = new Person;
            int nPopSize =
asForeignersPrimAgeSexProv[integer_age][sex][province_nat]->Count();
            if (nPopSize > 0) prPerson =
asForeignersPrimAgeSexProv[integer_age][sex][province_nat]
                ->GetRandom(RandUniform(23));
            else
            {
                nPopSize = asResidentsPrimAgeSex[integer_age][sex]->Count();
                if (nPopSize > 0) prPerson =
asResidentsPrimAgeSex[integer_age][sex]
                    ->GetRandom(RandUniform(24));
            }
            // a donor was found: assign donor's primary entry level
            if (nPopSize > 0 && prPerson->primary_level == PL_GRAD) primary_level
= PL_GRAD;
            // no donor was found; depending on samplpe size this should never or
rarely happen
            else if (RandUniform(25) < 0.5) primary_level = PL_GRAD;
        }
    }
}
}
}

```

```
}

```

Supporting functions

```

/* NOTE(PrimarySchool.AdjustedProbability,EN)
   This function modifies a base probability by two log odds factors. It is used to
   apply
   log odds by mothers education and a calibration factor to the base probability
*/
double PrimarySchool::AdjustedProbability(double dProb, double dLogOddEduc, double
dLogOddAdjust)
{
    if (dProb >= 0.9999) dProb = 0.9999;
    double dExp = exp(log(dProb / (1 - dProb)) + dLogOddEduc + dLogOddAdjust);
    return dExp / (1 + dExp);
}

/* NOTE(Person.DecidePrimaryEntry,EN)
   A simple function randomly setting the primary education entry status for a given
   probability
*/
void Person::DecidePrimaryEntry(double dProb)
{
    if (RandUniform(16) < dProb) primary_level = PL_ENTER;
}

/* NOTE(Person.DecidePrimaryGrad,EN)
   A simple function randomly setting the primary graduation status for a given
   probability
*/
void Person::DecidePrimaryGrad(double dProb)
{
    if (primary_level == PL_ENTER && RandUniform(17) < dProb) primary_level = PL_GRAD;
}

```

7.13.4. Table Output

```

table Person TabEducByYobAge15 //EN Education at age 15
[trigger_entrances(integer_age,15) ]
{
    sex + *
    province_birth+ *
    {
        unit
    }
    * year_of_birth
    * primary_level +
};

```

Changes in the Simulation() function in DYNAMIS-POP-MRT.mpp

The simulation engine now also has to create the primary school actor. The code is introduced after the person actors are created.

```

void Simulation()
{
    [...]
    for (long nJ = 0; nJ < TotalPopSampleSize; nJ++)

```

```

    {
        Person *paPerson = new Person();
        paPerson->Start(nJ, NULL);
    }

    // Create the Primary School actor
    PrimarySchool *paPrimarySchool = new PrimarySchool();
    paPrimarySchool->Start();
    [...]
}

```

Changes in PersonCore.mpp

We add a state for province of birth and a classification of provinces, including “abroad.”

```

classification PROVINCE_INT          //EN Province incl abroad
{
    PI_PROV00,                        //EN Hodh-Charghy
    PI_PROV01,                        //EN Hodh-Gharby
    PI_PROV02,                        //EN Assaba
    PI_PROV03,                        //EN Gorgol
    PI_PROV04,                        //EN Brakna
    PI_PROV05,                        //EN Trarza
    PI_PROV06,                        //EN Adrar
    PI_PROV07,                        //EN Dakhlett-Nouadibou
    PI_PROV08,                        //EN Tagant
    PI_PROV09,                        //EN Guidimagha
    PI_PROV10,                        //EN Tirs-Ezemour
    PI_PROV11,                        //EN Inchiri
    PI_PROV12,                        //EN Nouakchott
    PI_PROV13                          //EN Abroad
};

actor Person
{
    [...]
    PROVINCE_INT    province_birth = { PI_PROV00 };    //EN Province of birth
};

```

In the Start() function, we initialize province of birth of newborns by the province of residence of the mother.

```

void Person::Start(long PersonNr, Person *peMother)
{
    [...]
    province_nat = peMother->province_nat;
    province_birth = (PROVINCE_INT)peMother->province_nat;
    age = 0;
    time = time_of_birth;
    [...]
}

```

Changes in SatrtPopFile.mpp

We add the two new characteristics, province of birth and primary school level, from the starting population used at this step:

```

void Person::GetStartCharacteristics()
{

```

```

[...]
province_birth = (PROVINCE_INT) (int) prObservation->pmc[PMC_POB];
primary_level = (PRIMARY_LEVEL) (int) prObservation->pmc[PMC_EDUC];
[...]
```

Changes in the Immigration.mpp module

The province of birth is added as a characteristic at immigration:

```

void Person::GetImmigrantCharacteristics()
{
    [...]
    province_birth = (PROVINCE_INT) (SIZE (PROVINCE_INT) - 1);    // abroad is last
    category
}
```

7.14. Step 14: First Union Formation

7.14.1. Overview

This module implements first union formation (marriage) of women. It allows the user to choose between two approaches. The first option is applying a Coale & McNeil model. This allows parameterizing union formation by three parameters:

- Youngest age
- Average age of union formation
- Proportion of the population ever entering a union

Alternatively, users can specify period rates of age-specific union formation risks.

7.14.2. Concepts

This step provides an example of allowing users to choose the model to be applied to the modeled process.

Allowing users to choose between models

The union formation module allows the user to choose between two modeling options. Based on this choice, parameters are calculated and copied into a model-generated parameter, which is then applied independent of the user's choice for the rest of the code.

```

classification UNION_MODEL_SELECTION          //EN Model selection
{
    UMS_COALE,                                //EN Coale & McNeil
    UMS_RATES                                 //EN Rates by age, period and education
};

parameters
{
    //EN Model selection for union formation
    UNION_MODEL_SELECTION UnionModelSelection;

    //EN Union formation (Option A): Coale and McNeil
    double UnionParameters[PRIMARY_LEVEL][YOB_UNION][UNION_PARA];
```

```

//EN Union formation (Option B): Hazards
double UnionFormationHazardParameter[PRIMARY_LEVEL][AGE_UNION][SIM_YEAR_RANGE];

//EN Union formation hazards
model_generated double
UnionFormationHazard[PRIMARY_LEVEL][SIM_YEAR_RANGE][AGE_UNION];
};

/*
   In the presimulation of the union formation module, age specific hazards for
   union formation
   are calculated and copied into a model-generated parameter. Depending on the
   model selected by the user,
   values are calculated using a Coale McNeil model, or come directly from a
   parameter table.
*/
void PreSimulation()
{
    for (int nEduc = 0; nEduc < SIZE(PRIMARY_LEVEL); nEduc++)
    {
        for (int nYear = MIN(SIM_YEAR_RANGE); nYear <= MAX(SIM_YEAR_RANGE); nYear++)
        {
            for (int nAge = MIN(AGE_UNION); nAge <= MAX(AGE_UNION); nAge++)
            {
                if (UnionModelSelection == UMS_COALE)
                {
UnionFormationHazard[nEduc][RANGE_POS(SIM_YEAR_RANGE, nYear)][RANGE_POS(AGE_UNION, nAge)]
                    = GetUnionFormationHazard(nEduc, nYear, nAge);
                }
                else
                {
UnionFormationHazard[nEduc][RANGE_POS(SIM_YEAR_RANGE, nYear)][RANGE_POS(AGE_UNION, nAge)]
                    = UnionFormationHazardParameter[nEduc][RANGE_POS(AGE_UNION, nAge)]
                      [RANGE_POS(SIM_YEAR_RANGE, nYear)];
                }
            }
        }
    }
}

```

7.14.3. How to reproduce this modeling step

At this step, we add a module UnionFormation.mpp. Changes in other modules are very minor.

The UnionFormation.mpp module

Actor set and type definitions

The module uses one actor set for females by age and education; it is used to sample union formation dates from the female population and this information is input for immigrants. The user selection between two model types is implemented via a classification of possible choices.

```

// Females by age and education
actor_set Person asFemaleAgeEduc[integer_age][primary_level] filter set_alive &&
sex==FEMALE;

range YOB_UNION { 1964, 2050 }; //EN Year of birth
range AGE_UNION { 10, 60 }; //EN Age

```

```

classification UNION_PARA                //EN Union formation parameters
{
    UP_MINAGE,                            //EN Age at which first 1% enter a union
    UP_AVERAGE,                          //EN Average age at first union formation
    UP_EVER                                //EN Proportion ow femailes ever entering
    a union
};

classification UNION_MODEL_SELECTION     //EN Model selection
{
    UMS_COALE,                            //EN Coale & McNeil
    UMS_RATES                             //EN Rates by age, period and education
};

```

Model parameters

There are four parameters: the first for choosing the model to be used, one for each model, and a model-generated parameter in which the final parameters calculated from the chosen model parameters are copied in the pre-simulation.

```

parameters
{
    //EN Model selection for union formation
    UNION_MODEL_SELECTION UnionModelSelection;

    //EN Union formation (Option A): Coale and McNeil
    double UnionParameters[PRIMARY_LEVEL][YOB_UNION][UNION_PARA];

    //EN Union formation (Option B): Hazards
    double UnionFormationHazardParameter[PRIMARY_LEVEL][AGE_UNION][SIM_YEAR_RANGE];

    //EN Union formation hazards
    model_generated double
    UnionFormationHazard[PRIMARY_LEVEL][SIM_YEAR_RANGE][AGE_UNION];
};

parameter_group PG_UnionFormation      //EN Union formation
{
    UnionModelSelection, UnionParameters,
    UnionFormationHazardParameter
};

```

Actor states and functions

The key state modeled in this module is `in_union`, indicating that a woman has entered a union. The date can be recorded in the starting pulation file, modeled by the selected modeling choice, or input, in the case of immigrants.

```

actor Person
{
    AGE_UNION    age_union = COERCE(AGE_UNION, integer_age);        //EN Age

    //EN Union formation hazard
    double      union_formation_hazard = (WITHIN(AGE_UNION, integer_age) &&
sex==FEMALE) ?
    UnionFormationHazard[primary_level][RANGE_POS(SIM_YEAR_RANGE,
calendar_year)]
    [RANGE_POS(AGE_UNION, integer_age)] : 0.0;

    TIME        time_of_union_formation = { TIME_INFINITE };      //EN Time of

```

```

union formation
    logical    in_union = { FALSE };                //EN Ever in
union
    event      timeUnionFormationEvent, UnionFormationEvent;    //EN First union
formation event
    void        SetImmigrantUnionFormation();        //EN Union
formation date for immigrants
};

```

Calculating hazards using a Coale McNeil model

GetUnionFormationHazard() returns the union formation hazard for given education, age, and period based on a Coale McNeil model. This is a parametric model with three parameters: youngest age, average age of union formation, and proportion of the population ever entering a union.

This information is provided by a model parameter "UnionParameters."

```

double GetUnionFormationHazard(int nEduc, int nYear, int nAge);
double GetUnionFormationHazard(int nEduc, int nYear, int nAge)
{
    double dReturnValue = 0.0;
    double g[SIZE(AGE_UNION)];
    double a0 = UnionParameters[nEduc][RANGE_POS(YOB_UNION, nYear -
nAge)][UP_MINAGE];
    double my = UnionParameters[nEduc][RANGE_POS(YOB_UNION, nYear -
nAge)][UP_AVERAGE];
    double C = UnionParameters[nEduc][RANGE_POS(YOB_UNION, nYear - nAge)][UP_EVER];
    double k = (my - a0) / 11.36;
    double dThisYearCumulative = 0.0;
    if (nAge < a0) dReturnValue = 0.0;
    else if ( nAge >= a0 )
    {
        for (int nX = 0; nX < SIZE(AGE_UNION); nX++)
        {
            g[nX] = 0.19465*exp(-0.174*(nX - 6.06) - exp(-0.288*(nX - 6.06)));
        }
        double index = ((double)nAge - a0) / k;
        int nIndex = int(index); // integer part
        double dIndex = index - nIndex; // after comma part
        double G = 0.0;
        for (int nI = 0; nI <= nIndex; nI++)
        {
            if (nI < SIZE(AGE_UNION)) G = G + g[nI];
        }
        if (nIndex < SIZE(AGE_UNION) - 1) G = G + dIndex*g[nIndex + 1];
        dThisYearCumulative = G * C;
    }
    if (nAge == a0) dReturnValue = dThisYearCumulative;
    else if ( nAge > a0 )
    {
        double index = ((double)nAge - a0 -1) / k;
        int nIndex = int(index); // integer part
        double dIndex = index - nIndex; // after comma part
        double G = 0.0;
        for (int nI = 0; nI <= nIndex; nI++)
        {
            if (nI < SIZE(AGE_UNION)) G = G + g[nI];
        }
        if (nIndex < SIZE(AGE_UNION) - 1) G = G + dIndex*g[nIndex + 1];
    }
}

```

```

        if (1.0 - G * C > 0.0001) dReturnValue = (dThisYearCumulative - G * C) / (1 -
G * C);
        else dReturnValue = 0.0;
    }
    // convert probability to hazard
    if (dReturnValue < 1.0) dReturnValue = -log(1 - dReturnValue);
    return dReturnValue;
}

```

The union formation event

```

TIME Person::timeUnionFormationEvent()
{
    if (calendar_year >= MIN(SIM_YEAR_RANGE) & sex == FEMALE && !in_union &&
WITHIN(AGE_UNION, integer_age)
        && union_formation_hazard > 0.0)
    {
        return WAIT(-TIME(log(RandUniform(26)) / union_formation_hazard));
    }
    else if (time_of_union_formation < MIN(SIM_YEAR_RANGE) & sex == FEMALE
&& !in_union & calendar_year < MIN(SIM_YEAR_RANGE))
    {
        return time_of_union_formation;
    }
    else return TIME_INFINITE;
}

void Person::UnionFormationEvent()
{
    in_union = TRUE;
    time_of_union_formation = time;
}

```

Sampling the union formation age for immigrants

The function to sample a union formation age for immigrants based on age and primary education level is called at immigration. The host population is the total population of the same age and education.

```

void Person::SetImmigrantUnionFormation()
{
    if (person_type == PT_IMMIGRANT && sex==FEMALE &&
asFemaleAgeEduc[integer_age][primary_level]->Count()>0)
    {
        Person * prPerson = new Person;
        prPerson =
asFemaleAgeEduc[integer_age][primary_level]->GetRandom(RandUniform(27));
        time_of_union_formation = prPerson->time_of_union_formation;
        if (time_of_union_formation != TIME_INFINITE) in_union = TRUE;
    }
}

```

Presimulation

In the presimulation of the union formation module, age-specific hazards for union formation are calculated and copied into a model-generated parameter. Depending on the model selected by the user, values are calculated using a Coale McNeil model, or come directly from a parameter table.

```

void PreSimulation()

```

```

{
  for (int nEduc = 0; nEduc < SIZE(PRIMARY_LEVEL); nEduc++)
  {
    for (int nYear = MIN(SIM_YEAR_RANGE); nYear <= MAX(SIM_YEAR_RANGE); nYear++)
    {
      for (int nAge = MIN(AGE_UNION); nAge <= MAX(AGE_UNION); nAge++)
      {
        if (UnionModelSelection == UMS_COALE)
        {
          UnionFormationHazard[nEduc][RANGE_POS(SIM_YEAR_RANGE, nYear)][RANGE_POS(AGE_UNION, nAge)]
            = GetUnionFormationHazard(nEduc, nYear, nAge);
        }
        else
        {
          UnionFormationHazard[nEduc][RANGE_POS(SIM_YEAR_RANGE, nYear)][RANGE_POS(AGE_UNION, nAge)]
            = UnionFormationHazardParameter[nEduc][RANGE_POS(AGE_UNION, nAge)]
              [RANGE_POS(SIM_YEAR_RANGE, nYear)];
        }
      }
    }
  }
}

```

Tables

At this step, we add one table, which enables display of the calculated hazards and compares them with the simulated hazards. Besides hazards, the table also displays the proportion of women who ever entered a union. The table is by calendar year, age, and education.

```

table_group TG_05_Union_Tables //EN Union Formation
{
  UnionFormationTab
};

actor Person
{
  SIM_YEAR_RANGE tab_sim_year = COERCE(SIM_YEAR_RANGE, calendar_year); //EN Year
};

table Person UnionFormationTab //EN Union formation
[WITHIN(AGE_UNION, integer_age) && sex == FEMALE && WITHIN(SIM_YEAR_RANGE,
calendar_year)]
{
  {
    //EN Hazard from parameters decimals=4
    max_value_in(union_formation_hazard),

    //EN Simulated hazard decimals=4
    transitions(in_union, FALSE, TRUE) / duration(in_union, FALSE),

    //EN Proportion of women who ever entered a union decimals=4
    duration(in_union, TRUE) / duration()
  }
  * primary_level+
  * age_union
  * tab_sim_year
};

```

Changes in SetAliveEvent()

The sampling function for inputting a union formation date for immigrants at the moment of immigration is called in the “SetAliveEvent” and is called immediately after the actor is created (which, for immigrants, is the moment of immigration). To use education as a characteristic, SetImmigrantPrimaryEducationStatus() is explicitly called before, and the function hook used so far has to be removed.

```
void Person::SetAliveEvent()
{
    set_alive = TRUE;
    if (person_type == PT_IMMIGRANT)
    {
        SetImmigrantPrimaryEducationStatus();
        SetImmigrantUnionFormation();
    }
}
```

Changes in GetStartCharacteristics()

When reading the starting population file, the initialization of the union formation date has to be added:

```
void Person::GetStartCharacteristics()
{
    [...]
    if (prObservation->pmc[PMC_UNION] < MIN(SIM_YEAR_RANGE))
    {
        time_of_union_formation = prObservation->pmc[PMC_UNION];
    }
    [...]
}
```

7.15. Step 15: Fertility refined

7.15.1. Overview

This step refines the fertility module by adding an alternative way of modeling births and four options for which model to use and if and how to align it. Basically, fertility is implemented in two ways:

- The macro approach: rates by age only. Fertility is based on age-specific fertility rates calculated from two parameters: an age distribution of fertility and the TFR for future years. Another parameter is the sex ratio.
- The micro approach: fertility is modeled by age, parity, time since last birth, education, and union status. In this implementation, fertility is modeled based on a more detailed model estimated from micro-data. Users can choose to align the outcome to the macro outcomes, either for total outcome (number of births) or for births by age.

7.15.2. Concepts

This step provides an example of using the self_scheduling_split() function to determine the time interval since an event. It also provides an example for model alignment, where the number of events is determined by one model (the “macro” model), producing the alignment targets, while the selection to whom the event applies is based on another model (a refined “micro” model to find an appropriate person with the shortest waiting time).

State duration: Time intervals since an event

In this example, the time interval since the previous birth event is used for calculating the duration baseline risk for higher-order births. Cut-offs for time intervals are the first year since the previous birth, then 3, 6, 9, and 12 years. A logical indicator state is set to first FALSE and then TRUE in each birth event to reset the duration clock. This indicator is then used to split up the duration since the last event using `self_scheduling_split`. To calculate the most recent duration in a state, the function `active_spell_duration()` is used to determine the time a state has been in a given state since the last state change.

```
partition DUR_TIME_IN_PARITY{ 1, 3, 6, 9, 12 }; //EN Duration episodes since last
  birth

actor Person
{
  //EN Indicator set to FALSE and then again TRUE in the birth event
  logical in_this_parity = { FALSE };

  //EN Time index since the last birth event
  int time_in_parity
    = self_scheduling_split(active_spell_duration(in_this_parity, TRUE),
  DUR_TIME_IN_PARITY);
};

void Person::BirthEvent()
{
  in_this_parity = FALSE; // reset indicator: parity will increase
  parity++; // increment parity
  in_this_parity = TRUE; // set indicator true again
}
```

Model Alignment: Using one model for scheduling events happening to persons selected by another model

It must be determined at each birth event if the birth applies to the actor herself (unaligned models) or if an appropriate mother for the birth still has to be found. The latter is used for aligned modes, in which the birth events are produced by the macro model to which output is aligned, while the birth itself is given by the potential mother with the shortest waiting time determined by the micro model.

In this example, the time function is used in two ways: the “normal” way, to schedule events, and by directly calling it, with an indicator state “`override_mode`” set to TRUE, to find the actor with the shortest waiting time.

```
//EN Actor set of potential mothers
actor_set Person asPotentialMothers filter is_potential_mother;

TIME Person::timeBirthEvent()
{
  double dEventTime = TIME_INFINITE;
  double dHazard = 0.0;

  if (is_potential_mother)
  {
    // Timing comes from macro model (incl. models aligned to macro numbers)
    if ( use_macro_model && !override_mode)
    {
      dHazard = [.. Calculation of macro model hazard ];
    }

    // Timing comes from micro model or waiting time needed from micro model
  }
}
```

```

        else
        {
            dHazard = [.. Calculation of micro model hazard ]
        }

        if (dHazard > 0.0) dEventTime = WAIT(-TIME(log(RandUniform(28)) / dHazard));
    }
    return dEventTime;
}

void Person::BirthEvent ()
{
    // event applies to individual without alignment
    if ( not_an_aligned_model )
    {
        // increase parity of this actor and create a baby linked to this actor
        parity++; // increment parity
        Person *peChild = new Person; // Create and point to a new actor
        peChild->Start(-1, this); // Call the Start() function of the new
actor
    }
    // aligned to macro-model: find women with shortest waing time to birth
    else
    {
        Person *peMother = NULL;
        Person *pePotentialMother = NULL;
        double dShortestWaitTime = TIME_INFINITE;

        // find an appropriate mother for this birth event
        // loop through all potential mothers and find the one with the shortest
        // waiting time by explicitly calling the waiting time function

        double nNumber = asPotentialMothers->Count();
        for (double nJ = 0; nJ < nNumber; nJ++)
        {
            pePotentialMother = asPotentialMothers->Item(nJ);

            // Set override_mode in order to get waiting time based on micro model
            // from time function timeBirthEvent()
            pePotentialMother->override_mode = TRUE;

            // calculate waiting time and store shortest
            double dCurrentWaitTime = pePotentialMother->timeBirthEvent() - time;
            if (dCurrentWaitTime < dShortestWaitTime)
            {
                peMother = pePotentialMother;
                dShortestWaitTime = dCurrentWaitTime;
            }
            // reset override_mode
            pePotentialMother->override_mode = FALSE;
        }

        // increase parity of the selected mother and create a baby linked to her
mother
        peMother->parity++; // increment parity
        Person *peChild = new Person; // Create and point to a new
actor
        peChild->Start(-1, peMother); // Call the Start() function of
the new actor
    }
}

```

```
}
}
```

7.15.3. How to reproduce this modeling step

- Most changes and additions made in this step affect the module Fertility.mpp.
- In StartPopFile.mpp, code has to be added to set the state's parity and time since last birth.
- For immigrants, parity and time since last birth have to be sampled. This code can be added after sampling the time of union formation using the same donor (function SetImmigrantUnionFormation() in UnionFormation.mpp).

The new Fertility.mpp module

Actor sets

Actor sets are used to loop through potential mothers to find the one with the shortest waiting time. This is used if model alignment is chosen. Because the user can choose if a model is aligned by number of births only or by the number of births by age, two actor sets are provided.

```
//EN Potential mothers
actor_set Person asPotentialMothers filter is_potential_mother;

//EN Potential mothers by age
actor_set Person asPotentialMothersByAge[fertile_age] filter is_potential_mother;
```

Dimensions

Most dimensions used in this module are added at this step. The following is the complete code of declarations of dimensions.

```
// Dimensions

range FERTILE_AGE_RANGE { 10, 49 }; //EN Fertile age range
range PARITY_RANGE { 0, 15 }; //EN Parity range
range PARITY_RANGE1 { 1, 15 }; //EN Parity range 1+
range PARITY_RANGE2 { 2, 15 }; //EN Parity range 2+

classification SELECTED_FERTILITY_MODEL //EN Fertility model options
{
    SFM_MACRO, //EN Macro model (age only)
    SFM_MICRO, //EN Micro model un-aligned
    SFM_ALIGNE_BIRTHS, //EN Micro model with aligned
    number of births
    SFM_ALIGNE_AGE //EN Micro model with aligned
    number of births by age
};

classification UNION_STATUS //EN Union Status
{
    US_NO, //EN Never entered a union
    US_YES //EN Entered a union
};

classification HIGHER_BIRTHS_PARA //EN Parameters for higher order
births
{
    HBP_PERIOD1, //EN 0-1 years after previous
    birth
};
```

```

    HBP_PERIOD2, //EN 1-3 years after previous
    birth
    HBP_PERIOD3, //EN 3-6 years after previous
    birth
    HBP_PERIOD4, //EN 6-9 years after previous
    birth
    HBP_PERIOD5, //EN 9-12 years after previous
    birth
    HBP_PERIOD6, //EN 12+ years after previous
    birth
    HBP_AGE35, //EN Age 35-39
    HBP_AGE40, //EN Age 40-44
    HBP_AGE45, //EN Age 45+
    HBP_EDUC1, //EN Entered primary school but
    dropped out
    HBP_EDUC2 //EN Graduated from primary
    school
};

partition BIRTH_AGE_PART{ 35, 40, 45 }; //EN Age Groups
partition DUR_TIME_IN_PARITY{ 1, 3, 6, 9, 12 }; //EN Duration episodes since last
    birth

```

Parameters

New parameters include a parameter to select between the four model options, as well as all the parameters for the second model type introduced; this type models fertility by parity and various other characteristics. For easier use, the parameters are hierarchically grouped. The following is the complete list of parameters including the parameters already used in the previous model steps.

```

parameters
{
    // Model selection

    SELECTED_FERTILITY_MODEL    selected_fertility_model; //EN
    Fertility model selection

    // Macro model parameters

    double    AgeSpecificFertility[FERTILE_AGE_RANGE][SIM_YEAR_RANGE]; //EN Age
    distribution of fertility
    double    TotalFertilityRate[SIM_YEAR_RANGE]; //EN Total
    fertility rate
    double    SexRatio[SIM_YEAR_RANGE]; //EN Sex
    ratio (male per 100 female)

    // Micro model parameters

    //EN First Births
    double
    FirstBirthRates[PRIMARY_LEVEL][UNION_STATUS][FERTILE_AGE_RANGE][PROVINCE_NAT];

    //EN Higher order births
    double    HigherOrderBirthsPara[HIGHER_BIRTHS_PARA][PARITY_RANGE2]; //EN Higher
    order births

    //EN Birth Trends
    double    BirthTrends[PARITY_RANGE1][SIM_YEAR_RANGE];

    // Model-generated parameters

```

```

//EN Age specific fertility rate
model_generated double
AgeSpecificFertilityRate[FERTILE_AGE_RANGE][SIM_YEAR_RANGE];
};

//EN Fertility
parameter_group PG03_Fertility_Top
{
    selected_fertility_model, PG03a_Fertility_Model_A, PG03b_Fertility_Model_B
};

//EN Fertility Model A: Macro approach
parameter_group PG03a_Fertility_Model_A
{
    AgeSpecificFertility, TotalFertilityRate, SexRatio
};

//EN Fertility Model B: Micro approach
parameter_group PG03b_Fertility_Model_B
{
    FirstBirthRates, HigherOrderBirthsPara, BirthTrends
};

```

Actor Person declarations

A number of states is added at this step. The refined module allows the choice of a more detailed way of modeling fertility by accounting for parity, union status, and the duration since a previous birth. The way to get an automatically updated index of the time interval since last birth using the functions `self_scheduling_split()` and `active_spell_duration()` were explained in the section “Concepts” above. Unlike the example given there, in our model, we must also take care of the time since last birth recorded in the starting population file, which has to be added to the simulated time. As `self_scheduling_split()` only allows a set of functions as arguments (duration, weighted_duration, active_spell_duration, and active_spell_weighted_duration), we use the function `split()` instead. This function does not update its own event; instead, we ensure yearly updates by using an integer index of full years since last birth, which itself is a self-scheduling state.

The following code is the complete code of the actor `Person` block including states introduced at previous steps already.

```

actor Person
{
    FERTILE_AGE_RANGE    fertile_age = COERCE(FERTILE_AGE_RANGE, integer_age); //EN
Age
    int                  birth_age_part = split(integer_age, BIRTH_AGE_PART); //EN
Age group
    PARITY_RANGE         parity = { 0 }; //EN
Parity
    UNION_STATUS         union_status = (in_union) ? US_YES : US_NO; //EN
Union Status

    //EN Switch for calculating waiting times from one model while using another
    logical override_mode = { FALSE };

    //EN Indicator re-set at birth events for calculation of time since last birth
    logical in_this_parity = { TRUE };

    //EN Indicator that current year is in simulated time
    logical in_simulation_time = ( calendar_year >= MIN(SIM_YEAR_RANGE)) ? TRUE :

```

```

FALSE;

//EN Indicator re-set at birt events calculating time since last birth in
simulation time
logical in_this_parity_in_simulation = in_simulation_time && in_this_parity;

//EN Time since last birth in the starting population or at creation (immigrants)
double time_since_last_birth_start = { 0.0 };

//EN Time since last birth in full years
int time_since_last_birth =
self_scheduling_int(active_spell_duration(in_this_parity_in_simulation, TRUE))
+ round(time_since_last_birth_start);

//EN Time index since the last birth event
int time_in_parity = split(time_since_last_birth, DUR_TIME_IN_PARITY);

//EN Indicator that perion is a potential mother
logical is_potential_mother = (set_alive && sex == FEMALE &&
WITHIN(FERTILE_AGE_RANGE, integer_age)
&& parity < MAX(PARITY_RANGE) && WITHIN(SIM_YEAR_RANGE, calendar_year)) ?
TRUE : FALSE;

logical set_alive = { FALSE }; //EN Person is set to be alive
already
event timeSetAliveEvent, SetAliveEvent; //EN Event to set set_alive to
true
event timeBirthEvent, BirthEvent; //EN Birth event
};

```

The birth event

The time function now contains two ways of calculating waiting times, one following the previous “macro” model, and the second, a refined “micro” model introduced at this step. The time function is used in two ways: first, the “normal” way to schedule events, and second, explicitly for getting waiting times from the “micro” model (in `override_mode`) for finding the potential mother with the shortest waiting time. If the user chooses to align the model, the macro model is used for scheduling the event, but the birth happens to another person, i.e., the person with the shortest waiting time based on the micro model. Thus at each birth event it has to be determined if the birth applies to the actor herself (unaligned models) or if an appropriate mother for the birth still has to be found. The logic of this alignment method is explained in more detail above in the “Concepts” section.

At each birth we reset the indicator `in_this_parity` (setting it first `FALSE` and later `TRUE` again), which is used to calculate the state duration since last birth (using `active_spell_duration(in_this_parity,TRUE)`). Also the time since last birth (`time_since_last_birth_start`) from the starting population is reset to 0.

```

TIME Person::timeBirthEvent()
{
    double dEventTime = TIME_INFINITE;
    double dHazard = 0.0;

    if (is_potential_mother)
    {
        // Timing comes from macro model (incl. models aligned to macro numbers)
        if (selected_fertility_model != SFM_MICRO && !override_mode)
        {
            dHazard = AgeSpecificFertilityRate[RANGE_POS(FERTILE_AGE_RANGE,
integer_age)]
[RANGE_POS(SIM_YEAR_RANGE, calendar_year)];

```

```

    }

    // Timing comes from micro model or waiting time needed from micro model
    else
    {
        // first birth
        if (parity == 0)
        {
            dHazard =
FirstBirthRates[primary_level][union_status][RANGE_POS(FERTILE_AGE_RANGE,
integer_age)][province_nat]
            * BirthTrends[RANGE_POS(PARITY_RANGE1,
parity+1)][RANGE_POS(SIM_YEAR_RANGE, calendar_year)];
        }
        // higher order births
        else
        {
            double emil = time_since_last_birth;
            double hugo = 1.1;
            dHazard =
HigherOrderBirthsPara[time_in_parity][RANGE_POS(PARITY_RANGE2, parity+1)]
//Baseline
            * BirthTrends[RANGE_POS(PARITY_RANGE1,
parity+1)][RANGE_POS(SIM_YEAR_RANGE, calendar_year)]; //Trend
            integer gustav = time_in_parity;
            // relative risk for older age
            if (birth_age_part > 0) //not the baseline age group
            {
                dHazard = dHazard * HigherOrderBirthsPara[HBP_AGE35 - 1 +
birth_age_part][RANGE_POS(PARITY_RANGE2, parity+1)];
            }
            // relative risk for primary dropouts and graduates
            if (primary_level == PL_ENTER)
            {
                dHazard = dHazard *
HigherOrderBirthsPara[HBP_EDUC1][RANGE_POS(PARITY_RANGE2, parity+1)];
            }
            else if (primary_level == PL_GRAD)
            {
                dHazard = dHazard *
HigherOrderBirthsPara[HBP_EDUC2][RANGE_POS(PARITY_RANGE2, parity+1)];
            }
        }
    }

    // schedule event
    if (dHazard > 0.0) dEventTime = WAIT(-TIME(log(RandUniform(28)) / dHazard));
}
return dEventTime;
}

void Person::BirthEvent()
{
    // event applies to individual without alignment
    if (selected_fertility_model == SFM_MICRO || selected_fertility_model ==
SFM_MACRO)
    {
        // increase parity of this actor and create a baby linked to this actor
    }
}

```

```

        in_this_parity = FALSE;           // initialize indicator
        time_since_last_birth_start = 0.0; // reset time before start of simulation

        parity++;                         // increment parity
        Person *peChild = new Person;     // Create and point to a new actor
        peChild->Start(-1, this);         // Call the Start() function of the new
actor and pass own address
        in_this_parity = TRUE;           // reset indicator
    }
    // aligned to macro-model: find women with shortest waing time to birth
    else
    {
        Person *peMother = NULL;
        Person *pePotentialMother = NULL;

        double dShortestWaitTime = TIME_INFINITE;

        // align number of births only
        if ( selected_fertility_model == SFM_ALIGNE_BIRTHS )
        {
            double nNumber = asPotentialMothers->Count();
            // find an appropriate mother for this birth event
            for (double nJ = 0; nJ < nNumber; nJ++)
            {
                pePotentialMother = asPotentialMothers->Item(nJ);
                pePotentialMother->override_mode = TRUE;
                double dCurrentWaitTime = pePotentialMother->timeBirthEvent() - time;
                if (dCurrentWaitTime < dShortestWaitTime)
                {
                    peMother = pePotentialMother;
                    dShortestWaitTime = dCurrentWaitTime;
                }
                pePotentialMother->override_mode = FALSE;
            }
        }

        // align number of birth by age
        else
        {
            double nNumber = asPotentialMothersByAge[RANGE_POS(FERTILE_AGE_RANGE,
fertile_age)]->Count();
            // find an appropriate mother for this birth event
            for (double nJ = 0; nJ < nNumber; nJ++)
            {
                pePotentialMother =
asPotentialMothersByAge[RANGE_POS(FERTILE_AGE_RANGE, fertile_age)]->Item(nJ);
                pePotentialMother->override_mode = TRUE;
                double dCurrentWaitTime = pePotentialMother->timeBirthEvent() - time;
                if (dCurrentWaitTime < dShortestWaitTime)
                {
                    peMother = pePotentialMother;
                    dShortestWaitTime = dCurrentWaitTime;
                }
                pePotentialMother->override_mode = FALSE;
            }
        }

        // increase parity of the selected mother and create a baby linked to her

```

```

mother

    peMother->in_this_parity = FALSE;           // initialize indicator
    peMother->time_since_last_birth_start = 0.0; // reset time before start of
simulation

    peMother->parity++;                          // increment parity
    Person *peChild = new Person;                // Create and point to a new
actor
    peChild->Start(-1, peMother);                // Call the Start() function
of the new actor and pass own address
    peMother->in_this_parity = TRUE;             // reset indicator

}

```

Changes in StartPopFile.mpp

The following code for assigning values to the state's parity and time_since_last_birth is added in the function GetStartCharacteristics()

```

if ( sex == FEMALE )
{
    parity = (int)prObservation->pmc[PMC_PARITY];
    if ( parity > 0 && MIN(SIM_YEAR_RANGE) > prObservation->pmc[PMC_LASTBIR] )
    {
        time_since_last_birth_start = MIN(SIM_YEAR_RANGE) -
prObservation->pmc[PMC_LASTBIR];
    }
}

```

Changes in UnionFormation.mpp to assign values to immigrants

In the function SetImmigrantUnionFormation(), we add the following code to assign values to the state parity and time_since_last_birth_start using the same donor as for the date of first union formation.

```

parity = prPerson->parity;
time_since_last_birth_start = prPerson->time_since_last_birth;

```

7.16. Step 16: Infant mortality

7.16.1. Overview

This module implements a specific model for infant and child mortality and replaces the general mortality model for ages 0-4. The child mortality module is based on a proportional model consisting of:

- Mortality baseline hazards by age and sex
- Relative risks by age for mother's characteristics: age at birth (three groups), primary education (three groups)
- A time trend by age

The module is optional, and the user can choose between various options:

- Disable the child mortality model; the same model is used for all ages.

- Child mortality model without alignment: the child mortality module replaces the overall mortality module for ages 0-4. Note that the overall life expectancy may be altered by this choice.
- Child mortality model is calibrated for an initial year, then trends as for other ages: In this choice, life expectancy is the same as in the overall mortality for the given year, but as the composition of the population by mothers' characteristics changes over time, the number of deaths (and therefore life expectancy) will be different for the following years, allowing for scenario comparisons.
- Child mortality model calibrated for an initial year, then specific trends from the child mortality module: In this choice, life expectancy is the same as in the overall mortality for the given year, but as the composition of the population by mothers' characteristics changes over time, and as a result of potentially different trends, the number of deaths (and therefore life expectancy) will be different for the following years, allowing for scenario comparisons.

If activated, the model for child mortality starts replacing the overall mortality model five calendar years after the starting year of the simulation, ensuring that all children “know” their mother’s characteristics (are born in the simulation). As mother’s characteristics are not available for children born outside the country, immigrants are excluded from the model and handled by the general mortality module.

7.16.2. Concepts

From an implementation point of view, the main complication of this module lies in calibrating a model during execution. Five years into the simulation, all children subject to infant and child mortality are born in the simulation and thus know their mother’s characteristics. At this point, a calibration routine is performed, searching for baseline mortality rates by age and sex which, when applying to the relative risks by mothers’ characteristics, result in the same overall target mortality as the model without relative risks.

Calibrating a model during execution

The main problem of calibrating a model during execution is that model-generated parameters can only be set in the pre-simulation phase and not during execution. Calibration results thus cannot be stored as parameters, but have to be handled as states. To avoid storing these “parameters” as individual characteristics, a specific calibration actor can be defined and calibration results can be made states of this single actor. Individuals then can link to this actor and access the “parameter” values without having to store them individually.

The calibration actor

```
actor Calibrator
{
    double calibrated_value;           //EN Calibrated value
    event timeCalibrationEvent, CalibrationEvent; //EN Calibration event
};

TIME Calibrator::timeCalibrationEvent()
{
    .... return the time, when the calibration should be performed;
}

void Calibrator::CalibrationEvent()
{
    calibrated_value = calculate the value...
}
```

Accessing the calibrated value: A simple way to access the states of a single instance actor is by referencing it as the first item of an actor set:

```
actor_set Calibrator asCalibrator;

//... wherever the state is needed, it can be accessed as:
dLocalVariable = asCalibrator->Item(0)->calibrated_value;
```

Actors can also be permanently linked to the Calibrator and then access its states more directly:

```
// Links have to be defined, e.g.:

link Person.lCalibrator Calibrator.mlPerson[]; //EN Link between 1 calibrator and n
persons

//.. actors then can easily be linked, e.g. at the person's creation at Start()

lCalibrator = Calibrator->Item(0);

//.. to access states of the Calibrator, now the link can be used

dLocalVariable = lCalibrator->calibrated_value;
```

Actor linkages

Actors of the same or of different types can be linked, allowing access to the states of linked actors and easily interchanging information. Links can be of type 1:1 (e.g., between spouses), 1:n (e.g., children to mothers), and n:n (e.g., persons to siblings). Links have to be declared first. Once a link is established, it is maintained automatically and symmetrically, e.g., a child establishing a link to her mother automatically adds the link to her to the mother.

Declaration

```
// 1:1 of same actor type: link ActorName.LinkName
link Person.lSpouse;

// 1:1 of different actor types: link ActorOneName.LinkNameX ActorTwoName.LinkNameY
link Person.lSocialInsuranceAccount SocialInsuranceAccount.lPerson ;

// 1:n
link Person.lCalibrator Calibrator.mlPersons[];
link Person.lMother Person.mlChildren[];

// n:n
link Person.mlSportClubs[] SportClubs.mlMembers[];
```

Establishing a link A typical place to establish a link is in the Start() function, e.g., linking to a mother, if her address is passed over at birth through Start(Person *prMother).

```
// Declaration
link Person.lMother Person.mlChildren[]; // Link between mother and her children

// the mother creates a baby and calls the baby's start function passing over her own
address 'this'
...
prChild->Start(this);

// the baby permanently establishes a link
void Person::Start(Person *prMother)
{
...
}
```

```

    lMother = prMother;
}

// States of a linked actor can be accessed anywhere except for time functions of
// events.
// It is also possible to use links in derived states, for example:

logical    mother_is_alive = ( lMother != NULL ) ? TRUE : FALSE;
double    mothers_wealth = ( mother_is_alive ) ? lMother->wealth : 0.0;

```

Links are automatically maintained, so whenever a characteristic of an actor changes, this change becomes visible to the linked actor. Also, establishing or removing a link leads to the symmetric action on the other side of the linkage. A link is removed by setting it to NULL. For example, at divorce, the code `lSpouse=NULL` will remove the link on both ends of the previously married couple. Links are automatically removed at the death of an actor.

7.16.3. How to reproduce this step

Most of the changes are contained in the new module, `ChildMortality.mpp`, which is added at this step. Minor changes are required in the simulation engine, general mortality module, and `PersonCore` module.

The new module `ChildMortality.mpp`

Classifications and ranges

```

classification SELECTED_CHILD_MORTALITY_MODEL //EN Child Mortality Model Options
{
    SCMM_MACRO, //EN Disable child mortality model
    SCMM_MICRO_NOT_ALIGNED, //EN Child mortality model without
alignment
    SCMM_MICRO_ALIGNED_MACRO_TRENDS, //EN Aligned, trends as for other
ages
    SCMM_MICRO_ALIGNED_MICRO_TRENDS //EN Aligned, trends from child
mortality model
};

classification CHILD_MORTALITY_RISKS //EN Relative risks for child
mortality
{
    CMR_AGE14, //EN Mothers age at birth below 15
    CMR_AGE16, //EN Mothers age at birth below 17
    CMR_NOPRIM, //EN Mother never entered primary
education
    CMR_PRIMDROP //EN Mother dropped out of primary
education
};

classification MOTHER_AGE_CLASS //EN Mother age group
{
    MAC_14, //EN 14 and below
    MAC_16, //EN 15 or 16
    MAC_17P //EN 17 and above
};

range CHILD_MORTALITY_AGE { 0, 4 }; //EN Age range for child mortality
range CHILD_MORTALITY_YEARS{ 2018, 2113 }; //EN Year range for which child
mortality is modeled

```

Parameters

```

parameters
{
    //EN Child mortality model selection
    SELECTED_CHILD_MORTALITY_MODEL SelectedChildMortalityModel;

    //EN Child mortality baseline risk
    double ChildMortalityBaseRisk[CHILD_MORTALITY_AGE][SEX];

    //EN Child mortality relative risk
    double ChildMortalityRelativeRisks[CHILD_MORTALITY_AGE][CHILD_MORTALITY_RISKS];

    //EN Child mortality time trend
    double ChildMortalityTrend[CHILD_MORTALITY_AGE][CHILD_MORTALITY_YEARS];
};

parameter_group PG_ChildMortality //EN Child mortality
{
    SelectedChildMortalityModel, ChildMortalityBaseRisk,
    ChildMortalityRelativeRisks, ChildMortalityTrend
};

```

The *ChildMortalityCalibrator* actor declaration

To allow calibration of the baseline hazard of child mortality, we introduce a single instance calibration actor. This actor has a single event: the calibration of the model by finding baseline hazards that, when applying relative risks and for a given population composition determined at this moment of time, leads to the same number of deaths as using the base model. As parameters cannot be changed during simulation, the calibrated values are stored in states of the calibrator.

```

actor_set ChildMortalityCalibrator asTheChildMortalityCalibrator;
link Person.lChildMortalityCalibrator ChildMortalityCalibrator.mlBabies[];

actor ChildMortalityCalibrator
{
    double mort_male_0; //EN Child mortality risk male age 0
    double mort_male_1; //EN Child mortality risk male age 1
    double mort_male_2; //EN Child mortality risk male age 2
    double mort_male_3; //EN Child mortality risk male age 3
    double mort_male_4; //EN Child mortality risk male age 4

    double mort_female_0; //EN Child mortality risk female age
0
    double mort_female_1; //EN Child mortality risk female age
1
    double mort_female_2; //EN Child mortality risk female age
2
    double mort_female_3; //EN Child mortality risk female age
3
    double mort_female_4; //EN Child mortality risk female age
4

    logical calibration_is_done = { FALSE }; //EN Calibration is done
    void Start(); //EN Starts the calibrator actor

    //EN Child mortality calibration event
    event timeChildMortalityCalibrationEvent, ChildMortalityCalibrationEvent;

    integer int_age = self_scheduling_int(age);

```

```
};

void ChildMortalityCalibrator::Start()
{
    time = MIN(SIM_YEAR_RANGE);
}

```

Implementation of the calibration event

The event is called five years into projected time, when all children below five were born in the simulation and know their mother's characteristics, which are used to calculate relative risks in the model. The calibration is performed by a binary search algorithm. As part of the calibration, the population composition by age, sex, and the selected mother's characteristics is determined.

```
TIME ChildMortalityCalibrator::timeChildMortalityCalibrationEvent()
{
    // if not done yet and a model with alignment is chosen
    if (!calibration_is_done
        && ( SelectedChildMortalityModel == SCMM_MICRO_ALIGNED_MACRO_TRENDS
            || SelectedChildMortalityModel == SCMM_MICRO_ALIGNED_MICRO_TRENDS))
    // return the moment in which the calibration is to be performed
    {
        return MIN(CHILD_MORTALITY_YEARS);
    }
    // else never call the calibration event
    else return TIME_INFINITE;
}

void ChildMortalityCalibrator::ChildMortalityCalibrationEvent()
{
    // local matrices
    double dDeaths[SIZE(CHILD_MORTALITY_AGE)][SIZE(SEX)]; // Number expected
    deaths
    double dProb[SIZE(CHILD_MORTALITY_AGE)][SIZE(SEX)]; // Death probability
    double dBase[SIZE(CHILD_MORTALITY_AGE)][SIZE(SEX)]; // Baseline Hazard
    found in simulation
    double
    dRelRisks[SIZE(CHILD_MORTALITY_AGE)][SIZE(PRIMARY_LEVEL)][SIZE(MOTHER_AGE_CLASS)];
    // Relative risks
    long
    nPop[SIZE(CHILD_MORTALITY_AGE)][SIZE(SEX)][SIZE(PRIMARY_LEVEL)][SIZE(MOTHER_AGE_CLASS)]; //
    Pop sizes

    // Initialize matrices
    // set population sizes nPop and expected deaths nDeaths to 0
    // sets death probabilities dProb by age and sex for the year in which the model
    is calibrated
    for (int nAge = 0; nAge < SIZE(CHILD_MORTALITY_AGE); nAge++)
    {
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            dProb[nAge][nSex] = 1.0 - exp(-MortalityTable[nAge][nSex]
                * MortalityTrend[RANGE_POS(SIM_YEAR_RANGE,
                    MIN(CHILD_MORTALITY_YEARS))][nSex]);
            dDeaths[nAge][nSex] = 0.0;
            for (int nEduc = 0; nEduc < SIZE(PRIMARY_LEVEL); nEduc++)
            {
                for (int nMotherAge = 0; nMotherAge < SIZE(MOTHER_AGE_CLASS);
                    nMotherAge++)
                {

```

```

        nPop[nAge][nSex][nEduc][nMotherAge] = 0.0;
    }
}
}

// Initialize relative risks dRelRisks by age, mothers education, and mothers age
for (int nAge = 0; nAge < SIZE(CHILD_MORTALITY_AGE); nAge++)
{
    dRelRisks[nAge][PL_NO][MAC_14] =
ChildMortalityRelativeRisks[nAge][CMR_NOPRIM]
    * ChildMortalityRelativeRisks[nAge][CMR_AGE14];
    dRelRisks[nAge][PL_ENTER][MAC_14] =
ChildMortalityRelativeRisks[nAge][CMR_PRIMDROP]
    * ChildMortalityRelativeRisks[nAge][CMR_AGE14];
    dRelRisks[nAge][PL_GRAD][MAC_14] =
ChildMortalityRelativeRisks[nAge][CMR_AGE14];
    dRelRisks[nAge][PL_NO][MAC_16] =
ChildMortalityRelativeRisks[nAge][CMR_NOPRIM]
    * ChildMortalityRelativeRisks[nAge][CMR_AGE16];
    dRelRisks[nAge][PL_ENTER][MAC_16] =
ChildMortalityRelativeRisks[nAge][CMR_PRIMDROP]
    * ChildMortalityRelativeRisks[nAge][CMR_AGE16];
    dRelRisks[nAge][PL_GRAD][MAC_16] =
ChildMortalityRelativeRisks[nAge][CMR_AGE16];
    dRelRisks[nAge][PL_NO][MAC_17P] =
ChildMortalityRelativeRisks[nAge][CMR_NOPRIM];
    dRelRisks[nAge][PL_ENTER][MAC_17P] =
ChildMortalityRelativeRisks[nAge][CMR_PRIMDROP];
    dRelRisks[nAge][PL_GRAD][MAC_17P] = 1.0;
}

// Population Count
// calculates population sizes nPop by age, sex, mothers education, and mothers
age group
// calculates expected deaths dDeaths by age and sex
for (long nJ = 0; nJ < asAllPersons->Count(); nJ++)
{
    Person * prPerson = new Person();
    prPerson = asAllPersons->Item(nJ);
    if (prPerson->integer_age <= MAX(CHILD_MORTALITY_AGE) &&
prPerson->mother_known)
    {
        for (int nAge = 0; nAge < SIZE(CHILD_MORTALITY_AGE); nAge++)
        {
            for (int nSex = 0; nSex < SIZE(SEX); nSex++)
            {
                if (nSex == prPerson->sex && nAge == prPerson->integer_age)
                {
                    dDeaths[nAge][nSex] = dDeaths[nAge][nSex] +
dProb[nAge][nSex];
                    if (prPerson->mother_educ_0 == TRUE)
                    {
                        if (prPerson->mother_max_14 == TRUE)
nPop[nAge][nSex][PL_NO][MAC_14]++;
                        else if (prPerson->mother_max_16 == TRUE)
nPop[nAge][nSex][PL_NO][MAC_16]++;
                        else nPop[nAge][nSex][PL_NO][MAC_17P]++;
                    }
                }
            }
        }
    }
}

```

```

        else if (prPerson->mother_educ_1 == TRUE)
        {
            if (prPerson->mother_max_14 == TRUE)
nPop[nAge][nSex][PL_ENTER][MAC_14]++;
            else if (prPerson->mother_max_16 == TRUE)
nPop[nAge][nSex][PL_ENTER][MAC_16]++;
            else nPop[nAge][nSex][PL_ENTER][MAC_17P]++;
        }
        else
        {
            if (prPerson->mother_max_14 == TRUE)
nPop[nAge][nSex][PL_GRAD][MAC_14]++;
            else if (prPerson->mother_max_16 == TRUE)
nPop[nAge][nSex][PL_GRAD][MAC_16]++;
            else nPop[nAge][nSex][PL_GRAD][MAC_17P]++;
        }
    }
}

// find calibrated baselines
for (int nAge = 0; nAge < SIZE(CHILD_MORTALITY_AGE); nAge++)
{
    for (int nSex = 0; nSex < SIZE(SEX); nSex++)
    {
        double dUpper = 2.0;
        double dLower = 0.0;
        double dCenter = 1.0;
        double dNumberDeaths = 0.0;
        int nIterations = 10000;

        while ( abs(dNumberDeaths - dDeaths[nAge][nSex]) > 0.0001 && nIterations
> 0 )
        {
            nIterations--;
            dBase[nAge][nSex] = ( dLower + dUpper) / 2.0;

            dNumberDeaths = 0.0;

            //Calculate numer of deaths for given dBase
            for (int nEduc = 0; nEduc < SIZE(PRIMARY_LEVEL); nEduc++)
            {
                for (int nMotherAge = 0; nMotherAge < SIZE(MOTHER_AGE_CLASS);
nMotherAge++)
                {
                    dNumberDeaths = dNumberDeaths +
nPop[nAge][nSex][nEduc][nMotherAge]
                    * (1-exp(-dBase[nAge][nSex] *
dRelRisks[nAge][nEduc][nMotherAge]));
                    double hugo = nPop[nAge][nSex][nEduc][nMotherAge];
                    double emil = dBase[nAge][nSex];
                    double gustav = dRelRisks[nAge][nEduc][nMotherAge];
                    double hannelore = dDeaths[nAge][nSex];
                    double anton = 0;
                }
            }
        }
    }
}

```

```

        // shrink search interval
        if ( dNumberDeaths > dDeaths[nAge][nSex]) dUpper = dBase[nAge][nSex];
        else dLower = dBase[nAge][nSex];
    }
    int hugo = nIterations;
}
}

// set states
mort_male_0 = dBase[0][MALE]; mort_female_0 = dBase[0][FEMALE];
mort_male_1 = dBase[1][MALE]; mort_female_1 = dBase[1][FEMALE];
mort_male_2 = dBase[2][MALE]; mort_female_2 = dBase[2][FEMALE];
mort_male_3 = dBase[3][MALE]; mort_female_3 = dBase[3][FEMALE];
mort_male_4 = dBase[4][MALE]; mort_female_4 = dBase[4][FEMALE];

calibration_is_done = TRUE;
}

```

The child mortality event of the actor Person

Individual child mortality rates are implemented as derived states, accessing the “parameters” found by the calibrator.

```

actor Person
{
    //EN Child mortality overrides general mortality module
    logical child_mortality_overrides = calendar_year >= MIN(CHILD_MORTALITY_YEARS)
        && SelectedChildMortalityModel != SCMM_MACRO
        && integer_age <= MAX(CHILD_MORTALITY_AGE)
        && mother_known;

    logical mother_known = { FALSE }; //EN Mother is known
    logical mother_educ_0 = { FALSE }; //EN Mother never entered
primary school
    logical mother_educ_1 = { FALSE }; //EN Mother dropped out of
primary school
    logical mother_max_14 = { FALSE }; //EN Mother was 14 or below
at birth
    logical mother_max_16 = { FALSE }; //EN Mother was 15 or 16 at
birth

    //EN Baseline mortality
    double child_mortality = (lChildMortalityCalibrator == NULL) ? 0.0 :
        (integer_age == 0 && sex == MALE) ? lChildMortalityCalibrator->mort_male_0 :
        (integer_age == 1 && sex == MALE) ? lChildMortalityCalibrator->mort_male_1 :
        (integer_age == 2 && sex == MALE) ? lChildMortalityCalibrator->mort_male_2 :
        (integer_age == 3 && sex == MALE) ? lChildMortalityCalibrator->mort_male_3 :
        (integer_age == 4 && sex == MALE) ? lChildMortalityCalibrator->mort_male_4 :
        (integer_age == 0 && sex == FEMALE) ?
lChildMortalityCalibrator->mort_female_0 :
        (integer_age == 1 && sex == FEMALE) ?
lChildMortalityCalibrator->mort_female_1 :
        (integer_age == 2 && sex == FEMALE) ?
lChildMortalityCalibrator->mort_female_2 :
        (integer_age == 3 && sex == FEMALE) ?
lChildMortalityCalibrator->mort_female_3 :
        (integer_age == 4 && sex == FEMALE) ?
lChildMortalityCalibrator->mort_female_4 : 0.0;

    //EN Child mortality event

```

```

    event timeChildMortalityEvent, ChildMortalityEvent;
};

```

According to the user's model choices, the time function of the child mortality event determines waiting times for three alternative options. Note that the module can be deactivated, in which case no events are scheduled and the general model (in mortality.mpp) handles mortality.

```

TIME Person::timeChildMortalityEvent()
{
    double dEventTime = TIME_INFINITE;
    double dHazard = 0.0;

    if (child_mortality_overrides) // at risk (the model is used and overrides the
overall mortality)
    {
        // base risk * trend
        // unaligned model
        if (SelectedChildMortalityModel == SCMM_MICRO_NOT_ALIGNED)
        {
            dHazard = ChildMortalityBaseRisk[integer_age][sex]
                * ChildMortalityTrend[integer_age][RANGE_POS(CHILD_MORTALITY_YEARS,
calendar_year)];
        }

        // aligned with overall trend
        else if (SelectedChildMortalityModel == SCMM_MICRO_ALIGNED_MACRO_TRENDS)
        {
            dHazard = child_mortality
                * MortalityTrend[RANGE_POS(SIM_YEAR_RANGE, calendar_year)][sex]
                / MortalityTrend[RANGE_POS(SIM_YEAR_RANGE,
MIN(CHILD_MORTALITY_YEARS))][sex];
        }

        // aligned with specific child mortality trend
        else
        {
            dHazard = child_mortality
                * ChildMortalityTrend[integer_age][RANGE_POS(CHILD_MORTALITY_YEARS,
calendar_year)];
        }

        // relativ risks
        if (mother_educ_0) dHazard = dHazard *
ChildMortalityRelativeRisks[integer_age][CMR_NOPRIM];
        else if (mother_educ_1) dHazard = dHazard *
ChildMortalityRelativeRisks[integer_age][CMR_PRIMDROP];
        if (mother_max_14) dHazard = dHazard *
ChildMortalityRelativeRisks[integer_age][CMR_AGE14];
        else if (mother_max_16) dHazard = dHazard *
ChildMortalityRelativeRisks[integer_age][CMR_AGE16];

        // if hazard is positive calculate event time
        if (dHazard > 0) dEventTime = WAIT(-log(RandUniform(4)) / dHazard);
    }
    return dEventTime;
}

void Person::ChildMortalityEvent()
{

```

```

    MortalityEvent();
}

```

Changes in the simulation engine DYNAMIS-POP-MRT.mpp

The only change in the simulation engine is the creation of the calibration actor in the Simulation() function; the code can be added, e.g., after the primary school actor is created:

```

// Create the child mortality calibration actor
ChildMortalityCalibrator *paChildMortalityCalibrator = new
ChildMortalityCalibrator();
paChildMortalityCalibrator->Start();

```

Changes in the general mortality module Mortality.mpp

Assessing whether a person is at risk now also includes assessing whether mortality is overridden by the child mortality module.

```

TIME Person::timeMortalityEvent()
{
...
    // check if a person is at risk
    if (dMortalityHazard > 0.0 && in_projected_time && !child_mortality_overrides)
...
}

```

7.16.4. Changes in the PersonCore.mpp module

In the Start() function in PersonCore, additional code is needed to store mother's characteristics at birth. Also, a link is established to the calibration actor.

```

....
else if (person_type == PT_CHILD)
{
...
    // Mother's characteristics
    mother_known = TRUE; //Mother is known
    if (peMother->primary_level == PL_NO) mother_educ_0 = TRUE;
    else if (peMother->primary_level == PL_ENTER) mother_educ_1 = TRUE;
    if (peMother->integer_age <= 14) mother_max_14 = TRUE;
    else if (peMother->integer_age <= 16) mother_max_16 = TRUE;

    // establish a link to the calibrator actor
    lChildMortalityCalibrator = asTheChildMortalityCalibrator->Item(0);
}
....

```

7.17. Step 17: Migration refined

7.17.1. Overview

At this step, we add an alternative parameter for migration probabilities, adding the level of education as a dimension. The user can choose between the base and refined versions of the model.

7.17.2. Concepts

This is a simple step in which just two additional parameters and their roles in the time function of the migration event are added.

7.17.3. How to reproduce this step

At this step, two parameters are added, the first allowing the user to choose which migration parameters to use (i.e., with or without education as a dimension), and the second for migration probabilities by province, age group, and as the added dimension, education.

```
//EN Use parameters by education
logical ModelByEducation;

//EN Migration probability by education
double MigrationProbabilityEduc[PRIMARY_LEVEL][SEX][AGE5_PART][PROVINCE_NAT];
```

The time function of the migration event now checks which parameter to use:

```
// get the probability to move
double dMoveProb = 0.0;
if (ModelByEducation) dMoveProb =
MigrationProbabilityEduc[primary_level][sex][nAge5][province_nat];
else dMoveProb = MigrationProbability[sex][nAge5][province_nat];
```

7.18. Step 18: Extending Table and Micro-Data Output

7.18.1. Overview

At this step, we extend the micro-data output options and the list of variables that can be written to the output CSV file. Users can now choose which variables to include in the output, and output for various points in time. Users can choose the timeframe of outputs (e.g., 2015 - 2065) as well as the time interval between output events (e.g., every five years).

The second addition concerns the list of output tables. We add a broad variety of output tables whose output allows for a rich analysis of simulation results.

7.18.2. Concepts

The following paragraphs address adding a header line to CSV micro-data input and output files.

Adding a header line with variable names to csv files

Immediately after opening a CSV file for output, a header line can be written using a **write_header()** function. The string variable has to be of type `stc::string` or a compatible type like `CStringA` (the latter allowing for easy concatenating using the `+` operator; see example):

```
CStringA myString = "ID, ";           // Actor ID
myString = myString + "TIME, ";     // Time
...
out_csv.open(MicroRecordFileName);
out_csv.write_header(myString.GetString());
```

For reading a header file, the function is **read_header()**:

```
in_csv.open(PersonMicroDataFile);
std::string headerLine = in_csv.read_header();
```

7.18.3. How to reproduce this step

Extending the Micro-Data Output Options

Classifications and Parameters

New classifications are declared for the list of variables and for the extended output options. They are used to add or modify corresponding parameters.

```
classification OUTPUT_VARIABLES //EN List of output
variables
{
    OV_TIME, //EN Time
    OV_WEIGHT, //EN Weight
    OV_BIRTH, //EN Time of birth
    OV_SEX, //EN Sex
    OV_PROVINCE, //EN Province
    OV_EDUC, //EN Education
    OV_POB, //EN Province of
    birth
    OV_UNION, //EN Union formation
    time
    OV_PARITY, //EN Number of births
    OV_LASTBIR //EN Time of last
    birth
};

classification OUTPUT_TIMES //EN Micro-data
output times
{
    OT_FIRST, //EN Time of first
    output
    OT_LAST, //EN Time of last
    output
    OT_INTERVAL //EN Time interval (0
    for no repetition)
};

parameters
{
    ...
    double TimeMicroOutput[OUTPUT_TIMES]; //EN Time(s) of
    micro-data output
    logical SelectedOutput[OUTPUT_VARIABLES]; //EN Output variable
    selection
};
```

Presimulation

In presimulation, the output file is opened and a header line is written, containing the variable names:

```
void PreSimulation()
{
    if (WriteMicrodata)
    {
        CStringA myString = "ID, "; //
        Actor ID
```

```

        if (SelectedOutput [OV_TIME])          myString = myString + "TIME, ";          //
Time
        if (SelectedOutput [OV_WEIGHT])        myString = myString + "WEIGHT, ";        //
Weight
        if (SelectedOutput [OV_BIRTH])         myString = myString + "BIRTH, ";         //
Time of birth
        if (SelectedOutput [OV_SEX])           myString = myString + "MALE, ";           //
Sex
        if (SelectedOutput [OV_PROVINCE])       myString = myString + "PROVINCE, ";       //
Province
        if (SelectedOutput [OV_EDUC])          myString = myString + "EDUCATION, ";      //
Education
        if (SelectedOutput [OV_POB])           myString = myString + "PROV_BIRTH, ";      //
Province of birth
        if (SelectedOutput [OV_UNION])         myString = myString + "UNION, ";         //
Union formation time
        if (SelectedOutput [OV_PARITY])        myString = myString + "PARITY, ";        //
Number of births
        if (SelectedOutput [OV_LASTBIR])       myString = myString + "LAST_BIRTH, ";      //
time of last birth

        out_csv.open (MicroRecordFileName);
        out_csv.write_header (myString.GetString ());
    }
}

```

The output event

The output event is modified to allow output at various points in time (the timeinterval being a parameter) and to output the variables chosen by the user from the list of variables.

```

actor Person
{
    TIME    time_microdata_output = { TIME_INFINITE };    //EN Time for microdata
output
    void    WriteMicroRecord_Start ();                    //EN Initialization for
microdata output event
    hook    WriteMicroRecord_Start, Start;
    event   timeWriteMicroRecord, WriteMicroRecord;      //EN Write micro-data
record event
};

void Person::WriteMicroRecord_Start ()
{
    if (WriteMicrodata && TimeMicroOutput [OT_FIRST] >= time)
    {
        time_microdata_output = TimeMicroOutput [OT_FIRST];
    }
    else time_microdata_output = TIME_INFINITE;
}

TIME Person::timeWriteMicroRecord ()
{
    if (GetReplicate ()==0) return time_microdata_output;
    else return TIME_INFINITE;
}

void Person::WriteMicroRecord ()
{

```

```

// calculate variables
double dUnionFormation = -99;
if (sex == FEMALE && time_of_union_formation < time) dUnionFormation =
time_of_union_formation;
double dTimeLastBirth = -99;
if (sex == FEMALE && parity > 0) dTimeLastBirth = time_of_last_birth;

// Push the fields into the output record.
out_csv << actor_id; // Actor ID
if ( SelectedOutput[OV_TIME]) out_csv << time; // Time
if ( SelectedOutput[OV_WEIGHT]) out_csv << ActorWeight; // Weight
if ( SelectedOutput[OV_BIRTH]) out_csv << time_of_birth; // Time of
birth
if ( SelectedOutput[OV_SEX]) out_csv << (int)sex; // Sex
if ( SelectedOutput[OV_PROVINCE]) out_csv << (int)province_nat; // Province
if ( SelectedOutput[OV_EDUC]) out_csv << (int)primary_level; // Education
if ( SelectedOutput[OV_POB]) out_csv << (int)province_birth; // Province
of birth
if ( SelectedOutput[OV_UNION]) out_csv << dUnionFormation; // Union
formation time
if ( SelectedOutput[OV_PARITY]) out_csv << parity; // Number of
births
if (SelectedOutput[OV_LASTBIR]) out_csv << dTimeLastBirth; // time of
last birth

// All fields have been pushed, now write the record.
out_csv.write_record();

// set next output
if (time_microdata_output + TimeMicroOutput[OT_INTERVAL] > time &&
time_microdata_output + TimeMicroOutput[OT_INTERVAL] <=
TimeMicroOutput[OT_LAST])
{
time_microdata_output = time_microdata_output + TimeMicroOutput[OT_INTERVAL];
}
else
{
time_microdata_output = TIME_INFINITE;
}
}

```

Tables

At this step, we re-organize the whole table output of the model. The model produces a series of tables organized in seven table groups. All code is in the module Tables.mpp. The following discussion covers the entire code of the module and replaces the previous code.

Table groups

Population

- Total population by projected year, sex, and province
- Total population by projected year, age group, sex, and province
- Simulated starting population by age, sex, and province

Fertility

- Age-specific fertility rates by projected year

- Number of births by sex and projected year
- Number of births and first births by age group of mother and projected year
- Average age at birth and at first birth by education and projected year

Mortality

- Death rates and number of deaths by sex, age, and projected year
- Death rates and number of deaths by age group and projected year
- Child mortality by birth cohort and single year of age 0-4
- Mortality trends by sex (the trend factors calculated internally to scale the standard life table to meet the given scenario of future period life expectancy)

Migration

- Internal migration rates by age group and sex
- Number of immigrants by sex, age group, and province of destination by simulated year
- Emigration rates and number of emigrants by age group, sex, and province by projected year

Union

- First union formation by education, age, and simulated year: rates and proportion of women ever in a union
- Average age at first union formation by education and projected year

Education

- Population by province, age group, sex, and education by projected year
- Education composition of 15-year-old by sex, province of birth, and projected year
- Education composition of 15-year-old by sex, province of residence, and projected year

Female life-course experiences: cohort measures on own survival, union formation, education, fertility, and child deaths

- Average age at union formation
- Average age at first birth
- Proportion surviving until age 10
- Proportion surviving until age 20
- Proportion graduating from primary school
- Cohort fertility (subject to mortality)
- Cohort fertility of mothers (subject to mortality)
- Average number of children dying < age 5 per woman
- Average number of children dying < age 5 per mother

```
////////////////////////////////////  
// Table groups for organizing tables in the user interface  
//  
////////////////////////////////////
```

```

table_group TG01_PopulationTables //EN Population
{
    TabTotalPopulationSexProvince,
    TabPopulationAgeSexProvince,
    TabSimulatedStartingPopulation
};

table_group TG_02_FertilityTables //EN Fertility
{
    TabFertilityRates,
    TabNumberBirths,
    TabBirthsByAgeGroup,
    TabAgeAtBirth
};

table_group TG_03_MortalityTables //EN Mortality
{
    TabDeathRates,
    TabDeathsByAgeGroup,
    TabCohortChildMortality,
    TabMortalityTrendFactor
};

table_group TG_04_MigrationTables //EN Migration
{
    TabInternalMigrationRate,
    TabImmigrationSexProvinceAge,
    TabEmigration
};

table_group TG_05_UnionTables //EN First union
formation
{
    TabUnionFormation,
    TabAgeAtFirstUnionFormation
};

table_group TG_06_EducationTables //EN Education
{
    TabPopProvAgeEducSex,
    TabEduc15ByPob,
    TabEduc15ByPor
};

table_group TG_07_CohortTables //EN Cohort measures
{
    TabMothersExperience
};

```

States and dimensions used in tables

Some states are used exclusively in tables and are therefore defined here. If the tables are not used, these states can be deleted with the table code to free memory space. Most states (and also tables) belong to the actor Person. The exceptions are states and a table belonging to the actor ChildMortalityCalibrator. The child mortality calibrator actor has only one instance and thus can more efficiently implement a table with values that do not vary between persons (calibration results in our example).

The logical state count_parity is an indicator introduced to exclude the number of children assigned to newly created immigrants at immigration from birth counts. It becomes true almost immediately after an

actor is created and prevents inaccurate birth counts.

```

//////////////////////////////////////
// States and dimensions used in tables only (can be removed with tables)
//
//////////////////////////////////////

partition    LITTLE_TIME{ 0.001 };                               //EN A
very small time period
partition    AGE_15_17{ 15,17 };                                 //EN Age
groups
partition    AGE_MORT{ 1,5,15,65 };                             //EN Age
groups
range        SIM_YOB_RANGE{ 2013,2060 };                       //EN Year
of birth
range        CHILD_MORT_RANGE{ 0,14 };                         //EN
Child mortality age range

actor Person
{
    logical is_child = (integer_age < 15);                      //EN
Child < 15
    logical is_old = (integer_age >= 60);                       //EN
Person 60+
    logical is_aged0 = (!is_child && !is_old && primary_level == PL_NO); //EN
Person 15-59 never entered primary
    logical is_aged1 = (!is_child && !is_old && primary_level == PL_ENTER); //EN
Person 15-59 uncompleted primary
    logical is_aged2 = (!is_child && !is_old && primary_level == PL_GRAD); //EN
Person 15-59 primary graduate

    FERTILE_AGE_RANGE fert_age = COERCE(FERTILE_AGE_RANGE, integer_age); //EN Age
SIM_YEAR_RANGE      sim_yob = COERCE(SIM_YOB_RANGE, year_of_birth); //EN Year
of birth
    CHILD_MORT_RANGE  sim_cmr = COERCE(CHILD_MORT_RANGE, integer_age); //EN Age

    //EN Indicator to exclude the number of children assigned at immigration from
birth counts
    logical count_parity = self_scheduling_split(duration(set_alive, TRUE),
LITTLE_TIME);
};

actor ChildMortalityCalibrator
{
    //EN Calendar Year
SIM_YEAR_RANGE c_year = COERCE(SIM_YEAR_RANGE, self_scheduling_int(time));

    //EN Mortality trend calibration factor - male
double MaleMortalityTrend = MortalityTrend[RANGE_POS(SIM_YEAR_RANGE,
c_year)][MALE];

    //EN Mortality trend calibration factor - female
double FemaleMortalityTrend = MortalityTrend[RANGE_POS(SIM_YEAR_RANGE,
c_year)][FEMALE];
};

```

General Population Tables

```

//////////////////////////////////////

```



```

    {
        transitions(alive, TRUE, FALSE), //EN Number of persons
dying at this age
        transitions(alive, TRUE, FALSE) / duration() //EN Death Rate
decimals=4
    }
    * split(integer_age, AGE_MORT) + //EN Age group
    * sim_year //EN Calendar Year
};

table Person TabCohortChildMortality //EN Cohort child mortality rates
[in_projected_time && WITHIN(SIM_YOB_RANGE, year_of_birth) &&
WITHIN(CHILD_MORT_RANGE, integer_age)]
{
    {
        transitions(alive, TRUE, FALSE) / duration() //EN Mortality decimals=4
    }
    * sim_cmr //EN Age
    * sim_yob //EN Year of birth
};

table ChildMortalityCalibrator TabMortalityTrendFactor //EN Mortality
calibration factor
{
    c_year *
    {
        value_out(MaleMortalityTrend), //EN Male Mortality Trend
decimals=4
        max_value_out(FemaleMortalityTrend) //EN Female Mortality
Trend decimals=4
    }
};

```

Migration Tables

```

//////////////////////////////////////
// Migration Tables
//
//////////////////////////////////////

table Person TabInternalMigrationRate //EN Internal migration
rate
[ in_projected_time ]
{
    sex + *
    {
        number_moves / duration() //EN Migration rate decimals=4
    }
    * split(integer_age, AGE5_PART) //EN Age group
    * province_nat +
};

table Person TabImmigrationSexProvinceAge //EN Immigration by sex, province and
age group
[person_type == PT_IMMIGRANT && in_projected_time && trigger_entrances(set_alive,
TRUE)]
{
    sex + *
    province_nat + *

```

```

    {
        unit //EN Immigrants
    }
    * split(integer_age, AGE5_PART)+ //EN Age group
    * sim_year
};

table Person TabEmigration //EN Emigration rates and
numbers
[in_projected_time]
{
    sex + *
    province_nat + *
    {
        transitions(has_emigrated, FALSE, TRUE) / duration(), //EN Emigration
rates decimals=4
        transitions(has_emigrated, FALSE, TRUE) //EN Emigrants
    }
    * split(integer_age, AGE5_PART)+ //EN Age group
    * sim_year
};

```

Education Tables

```

//////////////////////////////////////
// Education Tables //
//////////////////////////////////////

table Person TabPopProvAgeEducSex //EN Population by province, age group, sex and
education
[in\_projected\_time]
{
    province\_nat + \*
    sex + \*
    {
        duration(is\_child, TRUE), //EN Children < 15
        duration(is\_aded0, TRUE), //EN Persons 15-59 never entered primary
school
        duration(is\_aded1, TRUE), //EN Persons 15-59 primary school
non-completer
        duration(is\_aded2, TRUE), //EN Persons 15-59 primary school graduate
        duration(is\_old, TRUE) //EN Persons 60+
    }
    \* sim\_year //EN Calendar Year
};

table Person TabEduc15ByPob //EN Education composition of 15 year old by province of
birth
[ integer\_age==15 && in\_projected\_time ]
{
    province\_birth + \*
    sex + \*
    {
        duration(primary\_level, PL\_NO) / duration(), //EN Never entered
primary school decimals=4
        duration(primary\_level, PL\_ENTER) / duration(), //EN Primary school
non-completer decimals=4
        duration(primary\_level, PL\_GRAD) / duration() //EN Primary school
graduate decimals=4
    }
};

```

```

    }
    \* sim\_year
};

table Person TabEduc15ByPor //EN Education composition of 15 year old by province of
residence
[integer\_age == 15 && in\_projected\_time]

{
  province\_nat + \*
  sex + \*
  {
    duration(primary\_level,PL\_NO) / duration(), //EN Never entered
primary school decimals=4
    duration(primary\_level,PL\_ENTER) / duration(), //EN Primary school
non-completer decimals=4
    duration(primary\_level,PL\_GRAD) / duration() //EN Primary school
graduate decimals=4
  }
  \* sim\_year
};

table Person TabEduc2015 //EN Population by education 2015
[calendar\_year == 2015]
{
  in\_capital + \*
  sex + \*
  {
    duration(primary\_level,PL\_NO), //EN Never entered primary school
duration(primary\_level,PL\_ENTER), //EN Primary school non-completer
duration(primary\_level,PL\_GRAD) //EN Primary school graduate
  }
  \*split(integer\_age, AGE\_FIVE) //EN Age group
};

table Person TabEduc2040 //EN Population by education 2040
[calendar\_year == 2040]
{
  in\_capital + \*
  sex + \*
  {
    duration(primary\_level,PL\_NO), //EN Never entered primary school
duration(primary\_level,PL\_ENTER), //EN Primary school non-completer
duration(primary\_level,PL\_GRAD) //EN Primary school graduate
  }
  \*split(integer\_age, AGE\_FIVE) //EN Age group
};

table Person TabEduc2065 //EN Population by education 2065
[calendar\_year == 2065]
{
  in\_capital + \*
  sex + \*
  {
    duration(primary\_level,PL\_NO), //EN Never entered primary school
duration(primary\_level,PL\_ENTER), //EN Primary school non-completer
duration(primary\_level,PL\_GRAD) //EN Primary school graduate
  }
  \*split(integer\_age, AGE\_FIVE) //EN Age group
};

```

Union Formation Tables

```

//////////////////////////////////////
// Union Tables
//
//////////////////////////////////////

table Person TabUnionFormation //EN First union formation by education and age
[WITHIN(AGE_UNION, integer_age) && sex == FEMALE && in_projected_time &&
count_parity]
{
  {
    //EN Hazard calculated from parameters decimals=4
    max_value_in(union_formation_hazard),

    //EN Simulated hazard decimals=4
    transitions(in_union, FALSE, TRUE) / duration(in_union, FALSE),

    //EN Proportion of women who ever entered a union decimals=4
    duration(in_union, TRUE) / duration()
  }
  * primary_level +
  * age_union
  * sim_year
};

table Person TabAgeAtFirstUnionFormation //EN Age at first union formation
[ sex == FEMALE && in_projected_time && count_parity ]
{
  {
    //EN Average age at first union formation decimals=2
    value_at_transitions(in_union, FALSE, TRUE, age) / transitions(in_union,
FALSE, TRUE)
  }
  * primary_level +
  * sim_year
};

```

Female Life-Course Experience Tables

```

//////////////////////////////////////
// Female lifecourse experiences
//
//////////////////////////////////////

table Person TabMothersExperience //EN Cohort measures of
fertility and child deaths
[ in_projected_time && WITHIN(SIM_YOB_RANGE, year_of_birth) && person_type ==
PT_CHILD && sex==FEMALE]
{
  {
    //EN Average age at union formation decimals=4
    value_at_transitions(in_union, FALSE, TRUE, age) / transitions(in_union,
FALSE, TRUE),

    //EN Average age at first birth decimals=4
    value_at_transitions(parity, 0, 1, age) / transitions(parity, 0, 1),

```

```
//EN Proportion surviving until age 10 decimals=4
transitions(integer_age, 9,10) / unit,

//EN Proportion surviving until age 20 decimals=4
transitions(integer_age, 14,15) / unit,

//EN Proportion surviving until age 20 decimals=4
transitions(integer_age, 19,20) / unit,

//EN Proportion graduating from primary school until age 20 decimals=4
entrances(primary_level, PL_GRAD) / unit,

//EN Cohort Fertility (subject to mortality) decimals=4
parity / unit,

//EN Cohort Fertility of mothers (subject to mortality) decimals=4
parity / transitions(parity,0,1),

//EN Average number of children dying < age 5 per woman decimals=4
infant_deaths / unit,

//EN Average number of children dying < age 5 per mother decimals=4
infant_deaths / transitions(parity,0,1)
}
* sim_yob //EN Year of birth
};
```

Chapter 8. Software Downloads

8.1. Model Users

8.1.1. Prerequisites

To run Modgen applications on a PC, the Modgen Prerequisites have to be installed. This program is free and can be downloaded from the Statistics Canada Website at: <http://www.statcan.gc.ca/eng/microsimulation/modgen/modgen>. Alternatively, the Modgen version used for the population projection model is Version 12.1.0.1

8.1.2. DYNAMIS-POP-MRT Phase 1 (User Version)

The Phase 1 version of the model is a micro-simulation implementation of the DemProj macro population projection model. It starts from a distributional table of people by age and sex and is non-confidential. No additional files are required to run the model, build scenarios, etc.

- DYNAMIS-POP-MRT-Phase1-User.zip

8.1.3. DYNAMIS-POP-MRT Phase 2 (User Version)

The Phase 2 version of the full model as developed alongside this report. It starts from a starting population file built from the census, which is confidential, and therefore, no starting population file is currently included in this package, but users can still open the model and study its parameter tables and built-in documentation. A synthetic starting population file that allows users to run simulations is under development and will be added when available.

- DYNAMIS-POP-MRT-Phase2-User.zip

8.2. Model Developers

8.2.1. Required Software for Modgen Developers

The model was implemented using **Modgen**, a generic micro-simulation programming language developed and maintained at Statistics Canada. Modgen is free and can be downloaded from: <http://www.statcan.gc.ca/eng/microsimulation/modgen/modgen>. This site also provides training and

information resources.

The Modgen version used for the population projection model is Version 12.1.0.1

Modgen 12.1. requires **Microsoft Visual Studio 15**. The free community version can be downloaded from the Microsoft site: <https://www.visualstudio.com/downloads/>. This software must be installed before Modgen is installed.

8.2.2. The DYNAMIS-POP-MRT model (full developer's version)

The complete suite of all model steps 1-18 is available here as download. Currently no starting population file, as required from Step 10 onward, is included in this package, as it is confidential. Developers still can open the model and study its code, however. A synthetic starting population file that allows users to run simulations is under development and will be added when available.

- DYNAMIS-POP-MRT-CompleteDownload.zip

