microWELT

Step-by-Step Implementation Guide

This document is the pdf-version of an online implementation guide available at:

https://www.microwelt.eu/Implementation/Implementation-Index.html

Version/retrieved: 2020-12-07

Implementation

MicroWELT is implemented in Modgen, a freely available generic microsimulation programming language developed and maintained by Statistics Canada. This report documents the implementation of microWELT step by step and allows to reproduce the model from scratch. Most of the steps correspond to a new module that was added in this step. The documentation includes a description of each module as well as the modelling code. The steps are organized in such a way that basic concepts of modgen programming are introduced first. This allows the use of this documentation as a textbook for the implementation of microsimulation models.

Contents:

Step 1: Creating a Population

The microWELT model development starts from a very generic basic template for typical continuous-time interacting population models. The model creates a starting population from a weighted file of observations. While not having much functionality at this point, the model provides a starting point 'hiding' some of the more tricky coding. Newcomers to Modgen are not required to fully understand all the code at this point: the following steps of model development introduce the basic concepts from a modelers perspective and provide step-by-step instructions for model building using Modgen.

Model Description

microWELT Step 1 constitutes the starting template of the model. It implements some basic functionality common to most interacting population models like file handling, the creation of a starting population, and the handling of weights to automatically scale all output to the total population size. The model creates a starting population from a weighted file of observations. Depending on record weights and the desired simulation size, a population of simulation actors of equal weights is created from the observations, whereby individual observations might or might not be picked or picked various times. The template also supports linking people to families or households. Other functionality includes the production of sample output tables and a module for micro-data output for generating simulated cross-section and panel data. Most of the code does not require any change as the model is developed. Exceptions are:

- The list of variables of the starting population file set in the ObservationCore file.
- The use of the starting population variables to initialize the states of the simulated actors in the Start() function contained in the PersonCore file.
- The list of states to be included in the model micro-data output file set in the MicroDataOutput file.

File input

The micro-data population **input file** is a csv-file which currently contains the following variables:

- Household ID
- Weight
- Time of birth
- Sex

- Role in family

The variables, including their order, have to correspond with the variables in the starting population file. The list can simply be expanded as required.

Parameters

The model currently has 7 **parameters** organized into 2 groups: Starting population and Microdata output.

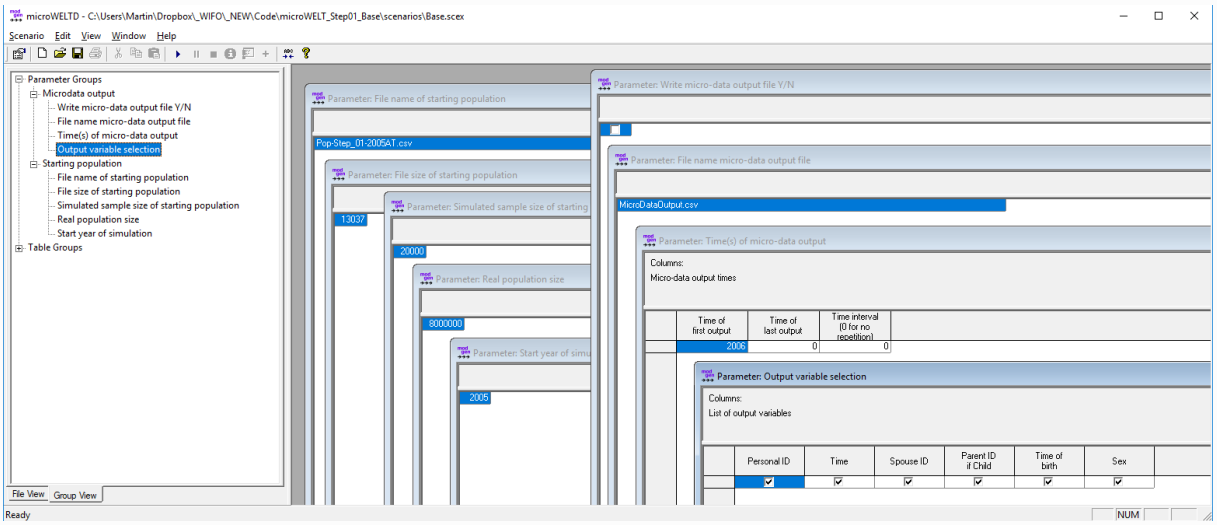

Figure: Model Parameters

The starting population parameters are:

- The file name
- The file size
- The simulated sample size (which can be bigger or smaller than the file)
- The real population size (to which all output is automatically scaled)

The starting population is generated from a weighted file of observations. Depending on weights and the desired simulation size, a population of actors is created from the observations, whereby individual observations might or might not be picked or picked various times.

Parameters for micro-data output are:

- A check-box to switch output on/off
- A file-name for the output file
- Parameters for the timing of the first and the last output and output intervals

Users can select when output should be written, which can be a single point in time or repeated panel waves with regular intervals.

Model Output

At this point, the model creates 3 output table as well as a micro-data output file. File output is generated in csv format including a header row containing variable names. Tables are part of the graphical user interface and there are various options to copy/paste or export tables.

Code organization and a classification of files

Modgen code is organized in .mpp files which can be classified into 4 groups.

Simulation engine: these files contain general model settings and the the code to run a simulation. Most of the code was automatically generated by the Modgen model wizard for time-based models. The code is mostly pure C++ and developers typically are not required to modify – or understand – it. The model is designed to run for various countries. All country and version specific code (like start year) are kept in a separate file _CountryContext.mpp.

- model_core.mp
- model_info.mpp
- modgen_timebased
- _CountryContext.mpp

Actor core files: Each Actor introduced in the model typically has a core file which contains general characteristics and functions of the actor. Our model at this point has four actor types, Observations (one for each record of the starting population file), Persons (the actual simulated entities representing the population), a Clock, and an actor Globals (for storing information easily accessible for all actors). Accordingly, there are four core files:

- ObservationCore.mpp
- PersonCore.mpp
- ClockCore.mpp
- GlobalsCore.mpp

Regular modules: These are the modules developers typically spend most of their time on. Usually there is one such module by modeled behavior (like fertility, mortality, labor force participation) and policy system. At the moment, our model contains just one such module:

- CalendarYearChange.mpp

Output modules: Modgen models produce three types of output: tables, micro-data, and a database of tracked actors used for a visualization tool called BioBrowser. The template provides two output files:

- Tables.mpp: contains all table output
- MicroDataOutput.mpp: handles micro-data output

Besides the model code, modes come with one or mode parameter files, as well as micro-data input and output files.

Parameter .dat files: values are stored in one or more .dat files, which are readable text-files. Parameters are typically changed within the graphical user interface of the model.

- Default(PersonCore).dat: contains all model parameters. The file-name consists of a scenario name (Default) and the parameter file name. A user creating and saving a new scenario would generate a new file, e.g. NewScenario(PersonCore).

Microdata csv input and output: Micro-data in/output is organized in csv (comma separated values) files, which are readable text files. File-names can be chosen by the model user, e.g.

- startpop_YYYY_COUNTRY.csv
- micro_output_YYYY_COUNTRY.csv

Model Documentation

Modgen models are self-documenting: Users can access a help file from the menu of the user interface.

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, thus good documentation is not only best practice for model development, but also creates a detailed documentation for the user.

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

Example:

```
//LABEL (CalendarYearChange, EN) Calendar Year Change

/*
   NOTE (CalendarYearChange, EN)
   This module handles calendar year changes. Before a year ends, the Person function YearEnd()
   is called by the Calendar clock actor. This is a point in time in which models typically
   update accounts and perform other end of year routines. Immediately after the year end, the
   YearStart() routine is called. This is when the calendar_year state is changed.
*/
```

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

```
//EN The text of the label
```

Labels are introduced by a language code (EN for English in our case) Modgen also support multilingual applications, in which these labels are translated into (an)other language(s). Besides their purpose of code documentation, these labels are used in the user interface, e.g. for column headings in tables or for the labeling of parameters.

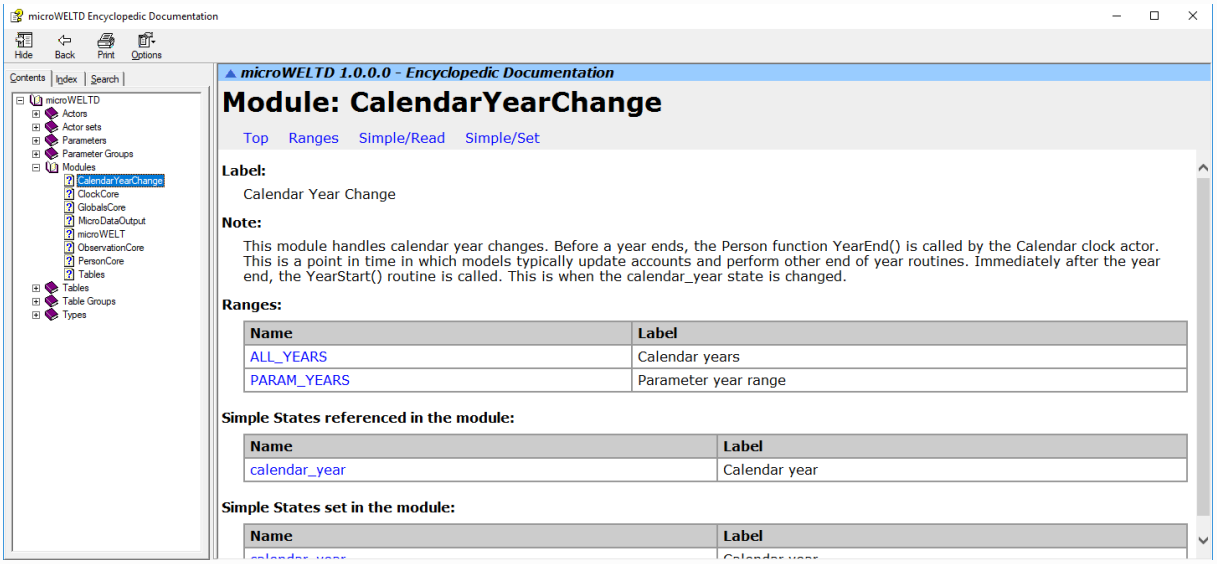

Figure: Help System

Code Discussion

Model Information: model_info.mpp

This file is used for model documentation and contains a model description. The file is for documentation only and does not contain any model code.

The Simulation Engine: model_core.mpp

This module contains general model settings and the simulation engine of the model. It is part of the model template and developers typically are not required to modify or understand it's code. Model-specific code includes:

- The creation of Observation actors reading the starting population file
- The calculation of integer weights corresponding with the selected number of simulated actors
- The creation of a starting population of equal weight Person actors
- The removal of all Observation actors as they are not used anymore after the starting population was built.

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// General Model Settings
////////////////////////////////////////////////////////////////////////////////////////////////
//

model_type time_based, just_in_time;        // The model type
options packing_level = 2;                  // Reduces memory use at the expense of speed
time_type double;                           // The data type used to represent time

// Other data types
real_type               float;
counter_type    ushort;
integer_type    short;
index_type              ulong;

languages                                   // Supported languages
{
    EN // English
};


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Simulation() is called by the simulation framework to simulate a replicate
////////////////////////////////////////////////////////////////////////////////////////////////
//

void Simulation()
{
    extern void LogSimulationStart();
```

```
extern void SimulateEvents();
LogSimulationStart();                   // Write simulation start message to log

////////////////////////////////////////////////////////////////////
// Start of the model-specific part of the simulation engine      //
////////////////////////////////////////////////////////////////////

// Create and open the input data file
input_csv in_csv;
in_csv.open(MicroDataInputFile);
in_csv.read_header();

// Create the Actor Globals
auto prGlobals = new Globals();
prGlobals->Start();

// Create the Clock
auto prClock = new Clock();
prClock->Start();

// Create observations
for (long nJ = 0; nJ < MicroDataInputFileSize; nJ++)
{
    in_csv.read_record(nJ);
    auto paObservation = new Observation();
    paObservation->Start(in_csv);
}

// Set sample weights (obs_weight) in the observations to represent the (integer) number of
// persons to be created out of each observation
double dSumWeights = 0.0;

for (long nJ = 0; nJ < MicroDataInputFileSize; nJ++)
{
    dSumWeights = dSumWeights + asObservationAll->Item(nJ)->pmc[PMC_WEIGHT];
}

for (long nJ = 0; nJ < MicroDataInputFileSize; nJ++)
{
    double  dWeight = asObservationAll->Item(nJ)->pmc[PMC_WEIGHT]
                * StartPopSampleSize / dSumWeights;
    int     nWeight = (int)dWeight;
    if (RandUniform(1) < dWeight - nWeight) nWeight++;
    asObservationAll->Item(nJ)->obs_weight = nWeight;
}

// Calculate exact person weight: A small correction as all family members eventually inherit
// the weight of the head which can be different from the integer weights assigned above
long SumSimActors = 0.0;
for (long nH = 0; nH < asObservationHeads->Count(); nH++)
{
    // Number family members
    int nFamSize = 1 + asObservationNonHeads[asObservationHeads->Item(nH)->fam_id]->Count();
    // sum of person weights in family with all having same weight as head
    SumSimActors = SumSimActors + nFamSize * asObservationHeads->Item(nH)->obs_weight;
}
asGlobals->Item(0)->person_weight = StartPopSize/ SumSimActors;

// Create the starting population
while (asObservationHeads->Count() > 0)
{
    // First create the head
    auto paObservation = asObservationHeads->Item(0);
    auto paPerson = new Person();
    paPerson->Start(paObservation, NULL);
    // And now all other members of this family
```

```
        for (int nJ = 0; nJ < asObservationNonHeads[paObservation->fam_id]->Count(); nJ++)
        {
            auto paOtherPerson = new Person();
            paOtherPerson->Start(asObservationNonHeads[paObservation->fam_id]->Item(nJ),
paPerson);
        }
        paObservation->obs_weight--;
    }

    // Delete all observation actors
    while (asObservationAll->Count() > 0) asObservationAll->Item(0)->Finish();

    // Close the microdata input file
    in_csv.close();

    ////////////////////////////////////////////////////////////////
    //   End of the model-specific part of the simulation engine      //
    ////////////////////////////////////////////////////////////////


    SimulateEvents();                       // Simulate events until there are no more.
}
```

Modgen Simulation Framework: modgen_time_based.mpp

This module implements core global functions for time-based models. It should not be necessary to modify any code in this module.

The Country/Version-Specific Settings: _CountryContext

This file contains the country and starting year specific code. This is the only file to be adapded when porting the model to a new country context.

```


// Country: Austria

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////////////
//

range ALL_YEAR_RANGE { 1899, 2150 };            //EN All calendar years
```

Core file of the Observation Actor: ObservationCore.mpp

This module introduces an actor named ‘Observation’. Each Observation actor corresponds with a record of the starting population file and is created by the simulation engine before Person actors are created. Observation are weighted. According to the size of the

simulated population selected by the user, the simulation engine creates integer-weights for each observation by rescaling the original weights and random-rounding. The new observation weights are then used by the simulation engine to decide if and how often an observation is used when creating Person actors (which clone the observation characteristics). After being used for creating the starting population, the observation actors are destroyed to free up teh memory space. Note that for this process, the weight of the household head is used for all members of a family.

This module is part of the model template and developers typically are not required to modify – or understand – it's code. The exception is adding new variables to the starting population file which is a frequent requirement in model development:

How to add a new variable to the starting population?

- Add a new column to the csv micro-data file
- in this module, extend the list of fields in the classification 'PERSON_MICRODATA_COLUMNS' no more coding is required and the variable is read in automatically
- the new variable will typically be used in the Start() function of the Person actor where a corresponding state is initialized by its value. It is accessible using the new dimension name added to the list of fields as an index of the person-micro-column array pmc[NEW_FIELD]. See Code in the Start() function in PersonCore.mpp

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Dimensions
////////////////////////////////////////////////////////////////////////////////////////////
//


// LABEL(PERSON_MICRODATA_COLUMNS, EN) List of Starting population variables
classification PERSON_MICRODATA_COLUMNS
{
    PMC_HHID,                                   //EN Household ID
    PMC_WEIGHT,                                 //EN Weight
    PMC_BIRTH,                                  //EN Time of birth
    PMC_SEX,                                    //EN Sex
    PMC_ROLE                                    //EN Role in family
};



range FAM_ID{ 0,220000 };                       //EN Range of Family IDs

////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    file    MicroDataInputFile;                 //EN File name of starting population
    long    MicroDataInputFileSize;             //EN File size of starting population
    double  StartPopSampleSize;                 //EN Simulated sample size of starting population
    double  StartPopSize;                       //EN Real population size
};

parameter_group PG_ModelSettings                //EN Starting population
{
    MicroDataInputFile, MicroDataInputFileSize,
    StartPopSampleSize, StartPopSize
};


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor-Sets
////////////////////////////////////////////////////////////////////////////////////////////////
//

//EN Actor-set of all family heads in Observations
actor_set Observation asObservationHeads filter fam_role == FR_HEAD && obs_weight > 0;

//EN Actor-set of all family members (without heatds) by family ID
actor_set Observation asObservationNonHeads[fam_id] filter fam_role != FR_HEAD;

//EN All observations
actor_set Observation asObservationAll;

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states and functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

/*  NOTE(Observation, EN)
    The Actor Observation is created as internal representation of the starting population file
    records. It is used to create Person actors of the starting population which can be smaller or
    larger as the starting population file. The weights of observations are used for determining
if
    and how often a single observation is represented in the starting population.
*/


actor Observation                               //EN Actor Observations
{
    double      pmc[PERSON_MICRODATA_COLUMNS];  //EN Person micro record columns
    integer     obs_weight = { 1 };             //EN Observation integer weight
    FAM_ID      fam_id = { 0 };                 //EN Family ID
    FAM_ROLE    fam_role = { FR_HEAD };         //EN Role in family

    void        Start(const input_csv& input);  //EN Function starting the actor
    void        Finish();                       //EN Function destroying the actor
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void Observation::Start(const input_csv& in_csv)
```

```
{
    for (int nJ = 0; nJ < SIZE(PERSON_MICRODATA_COLUMNS); nJ++)
    {
        pmc[nJ] = in_csv[nJ];
    }
    fam_id      = (int)pmc[PMC_HHID];
    fam_role    = (FAM_ROLE)(int)pmc[PMC_ROLE];
};

void Observation::Finish(){};
```

Core file of the Person Actor: PersonCore.mpp

This module introduces the main actor of the model: Person. It contains the core functionality and general states of the actor Person. Each actor typically has a 'Core' module for handling general functionalities of the actor. The most essential functions of this module are the Start() and Finish() functions of the actor. Start() is called when an actor is created and initializes all states. It is the only place where the automatically provided states age and time can be set.

As for any actor, its core file is a good place to declare states and corresponding types which do not belong to specific behaviors but are used by various modules (e.g. sex). Also, the Start() and Finish() functions of each actor are declared in its core file.

```


////////////////////////////////////////////////////////////////////////////////////////////
//
// Dimensions
////////////////////////////////////////////////////////////////////////////////////////////
//

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

classification PERSON_TYPE                      //EN Person Type
{
    PT_START,                                   //EN Person from Starting Population
    PT_CHILD,                                   //EN Person born in simulation
    PT_IMMIGRANT                                //EN Immigrant
};

classification FAM_ROLE                         //EN Role in family
{
    FR_HEAD,                                    //EN Head
    FR_SPOUSE,                                  //EN Spouse of head
    FR_CHILD                                    //EN Child
};

////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
// Links
////////////////////////////////////////////////////////////////////////////////////////////////
//

link Person.lParent Person.mlChild[];              //EN Link between Head and Children
link Person.lSpouse;                                  //EN Link between spouses

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor sets
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor_set Person asAllPersons filter is_alive;  //EN Entire population

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states and functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

/*      NOTE(Person, EN)
    The actors Person are the main actors of the simulation. All persons have the
    same weight and together represent the population.
*/

actor Person
{
    PERSON_TYPE     person_type = { PT_START };                    //EN Person type
    double          time_of_birth = { 1900 };                     //EN Time of birth
    ALL_YEAR_RANGE  year_of_birth = int(time_of_birth);           //EN Year of birth
    SEX             sex = { FEMALE };                             //EN Sex
    FAM_ROLE        family_role = { FR_HEAD };                    //EN Family Role

    void                    Start(Observation *peObservation, Person *pePers);  //EN Starts the
actor
        void                Finish();
//EN Finishes the actor

    logical         is_alive = { FALSE };                         //EN Person is alive
    double          time_set_alive = { TIME_INFINITE };           //EN Time setting actor alive
    event           timeSetAliveEvent, SetAliveEvent;             //EN Set Alive

    Person          *peHHead;                                     //EN Pointer to household head

};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

/*      NOTE(Person.Start, EN)
    The Start() function initializes all states of an actor right at the moment of creation.
    The function has two parameters, namely pointers to an observation actor and to the household
    head. If the pointer to an observation is not NULL, the Person comes from the starting
    population (rather than being born or immigrating in the simulation). The pointer is used to
    access all states of the observation. The pointer to the household head is used to establish a
    link to the family head which can be a parent or a spouse. If the parameter is NULL, the
Person
    is a family head herself.
*/

void Person::Start(Observation *peObservation, Person *pePers)
{
```

```
    // Setting the actor weight
    Set_actor_weight(asGlobals->Item(0)->person_weight);
    Set_actor_subsample_weight(asGlobals->Item(0)->person_weight);

    // Determine the person type
    if (peObservation == NULL && pePers != NULL ) person_type = PT_CHILD;   // Born in simulation
    else if (peObservation != NULL ) person_type = PT_START;                // From Starting Pop
    else person_type = PT_IMMIGRANT;                                        // Immigrant

    // Initialize states
    if (person_type == PT_START) // Person comes from starting population
    {
        // (A) States from Starting population file
        time                = peObservation->pmc[PMC_BIRTH] + RandUniform(2);
        sex                 = (SEX)(int)peObservation->pmc[PMC_SEX];
        family_role = ( FAM_ROLE )( int )peObservation->pmc[PMC_ROLE];

        // (B) Other states
        time_of_birth       = time;
        calendar_year       = (int)time_of_birth;

        // (C) Links to head resp. spouse
        if (pePers != NULL) peHHead = pePers;
    }
    else if (person_type == PT_CHILD) // Person born in simulation
    {
        // do nothing (births not modeled yet)
    }
    else // Person is an immigrant
    {
        // do nothing (immigrants not modeled yet)
    }
    time_set_alive = WAIT(0);
}

/*      NOTE(Person.Finish, EN)
    Finish is a function to be called when an actor is to be destroyed. Modgen automatically
    performs memory clean-up routines. Code added here is the last thing executed before an actor
    is destroyed. Typical uses are the handling of inheritances and accounting routines.
*/
void Person::Finish(){}


/*  NOTE(Person.SetAliveEvent, EN)
    This function is called with waiting time 0 immediately after the Start of a Person actor.
    For people of the starting population the date of birth is only known after getting this
    information from the corresponding person record, thus after the person actor is created.
    The SetAliveEvent Event ensures that no person is visible and counted before its actual
    birth date.
*/

TIME Person::timeSetAliveEvent() { return time_set_alive; }
void Person::SetAliveEvent()
{
    lClock          = asTheClock->Item(0);      // link the person to the clock
    lGlobals        = asGlobals->Item(0);       // link the person to globals

    is_alive        = TRUE;                     // set the Person alive
    time_set_alive = TIME_INFINITE;             // ensure the event does not happen again

    if (person_type == PT_START && family_role != FR_HEAD && peHHead != NULL)
    {
        if (family_role == FR_CHILD) lParent = peHHead;          // Person is a child
        else lSpouse = peHHead;                                  // Person is a spouse
    }
    else if (person_type == PT_CHILD) lParent = peHHead;
```

```
}
```

Core file of the Clock Actor: ClockCore.mpp

This module introduces a Clock actor which handles calendar clock events like year changes. This avoids inefficiencies as otherwise all actors would having to schedule their own events individually. Instead of that the Clock announces each year end/start to all other actors

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Links
////////////////////////////////////////////////////////////////////////////////////////////////
//

link Person.lClock Clock.mlPersons[];

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Clock Actor
////////////////////////////////////////////////////////////////////////////////////////////////
//

/*  NOTE (Clock, EN)
    The Clock actor handles calendar year and other regular period changes.
*/

actor Clock                                                 //EN Clock Actor
{
    void    Start();                                        //EN Start Clock
    void    Finish();                                       //EN Finish
    int     clock_year = { 2000 };                          //EN Calendar Year
    int     next_clock_year_end = { TIME_INFINITE };        //EN Next end year clock tick
    int     next_clock_year_start = { TIME_INFINITE };      //EN Next start year clock tick
    event   timeStartYearClock, StartYearClock;             //EN Start year function of clock
    event   timeEndYearClock, EndYearClock;                 //EN End year function of clock
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor sets
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor_set Clock asTheClock;                                 //EN Actor Set Clock Actor

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void Clock::Start()
{
    time = MIN(ALL_YEAR_RANGE);
    clock_year = MIN(ALL_YEAR_RANGE);
    next_clock_year_end = WAIT(1);
```

```
};

void Clock::Finish() {}

/*      NOTE( Clock.EndYearClock, EN)
    The function EndYearClock creates an end of year event. It calls all other actors to announce
    that the year has ended so that all actors can perform their own end of year routines
*/

TIME Clock::timeEndYearClock() { return next_clock_year_end; }
void Clock::EndYearClock()
{
    long nPopSize = asAllPersons->Count();
    for (long nIndex = 0; nIndex < nPopSize; nIndex++)
    {
        auto prPerson = asAllPersons->Item(nIndex);
        prPerson->YearEnd();
    }
    next_clock_year_start = WAIT(0);
    next_clock_year_end = WAIT(1);
}

/*      NOTE( Clock.StartYearClock, EN)
    The function StartYearClock creates a new year event. It calls all other actors to announce
    that a new year has started so that all actors can call their own new year routines
*/

TIME Clock::timeStartYearClock() { return next_clock_year_start; }
void Clock::StartYearClock()
{
    clock_year++;
    long nPopSize = asAllPersons->Count();
    for (long nIndex = 0; nIndex < nPopSize; nIndex++)
    {
        auto prPerson = asAllPersons->Item(nIndex);
        prPerson->YearStart();
    }
    next_clock_year_start = TIME_INFINITE;
}
```

Core file of the Globals Actor: GlobalsCore.mpp

This module introduces an actor Globals which handles global variables and constants which are calculated or changed after the pre-simulation phase. All persons are linked to the actor Globals and can access it states. This is used to store information which applies to all Person actors on a central place avoiding redundancies. This functionality is typically used by modules which run calibrations after the simulation has been started.

```


////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Sets and linkages
////////////////////////////////////////////////////////////////////////////////////////
//

actor_set Globals asGlobals;                    //EN Globals (Actor Set)
link Person.lGlobals Globals.mlPerson[];               //EN Link Person to to Globals
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor States
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Globals                                   //EN Actor Globals
{
    double person_weight = { 1.0 };             //EN Person weight
};
```

Actions at Year End: CalendarYearChange.mpp

This module handles calendar year changes. Before a year ends, the Person function YearEnd() is called by the Calendar clock actor. This is a point in time in which models typically update accounts and perform other end of year routines. Immediately after the year end, the YearStart() routine is called. This is when the calendar_year state is changed.

```


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Person states and functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    ALL_YEAR_RANGE  calendar_year = { 2000 };                   //EN Calendar year
    void            YearEnd();                                  //EN Year End Function
    void            YearStart();                                //EN Year Start Function
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

/*      NOTE(Person.YearEnd, EN)
    The function YearEnd is called by the Clock actor at the end of each year. The code in this
    function is executed last thing in a given year before the calendar year is incremented
*/

void Person::YearEnd()
{
    // empty for the moment
}

/*      NOTE(Person.YearEnd, EN)
    The function YearStart is called by the Clock actor at the start of each year.
    The code in this function is executed first thing in new year
*/
void Person::YearStart()
{
    calendar_year = lClock->clock_year;
}
```

Micro Data Output: MicroDataOutput.mpp

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. Output can also be produced in fixed time intervals, e.g. every year or every 5 years. The output csv file includes a header line with variable names. The module can be switched on and off.

How to add a new variable to the output?

- In WriteMicroRecord(), add a line which pushes a new variable into the output record
- In PreSimulation() add a line to extend the list of variable names written to the file

```
output_csv out_csv;                                 //EN Microdata output csv object

////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////
//

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
////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    logical            WriteMicrodata;                  //EN Write micro-data output file Y/N
    double             TimeMicroOutput[OUTPUT_TIMES];   //EN Time(s) of micro-data output
    file               MicroRecordFileName;             //EN File name micro-data output file
};

parameter_group PG05_Files                          //EN Microdata output
{
    WriteMicrodata, MicroRecordFileName,
    TimeMicroOutput
};

////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states, functions and events
////////////////////////////////////////////////////////////////////////////////////////
//

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;            //EN Function Hook
    event timeWriteMicroRecord, WriteMicroRecord;   //EN Write micro-data record event
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////
//

void Person::WriteMicroRecord_Start()
{
    double dFirstOutput = TimeMicroOutput[OT_FIRST];
    while (WriteMicrodata && dFirstOutput <= TimeMicroOutput[OT_LAST] && dFirstOutput < time)
    {
        dFirstOutput = dFirstOutput + TimeMicroOutput[OT_INTERVAL];
        if ( TimeMicroOutput[OT_INTERVAL] <= 0 ) dFirstOutput = TIME_INFINITE;
    }

    if (WriteMicrodata && dFirstOutput >= time && dFirstOutput <= TimeMicroOutput[OT_LAST])
    {
        time_microdata_output = dFirstOutput;
    }
    else time_microdata_output = TIME_INFINITE;
}

TIME Person::timeWriteMicroRecord()
{
    if (GetReplicate()==0)  return time_microdata_output;
    else return TIME_INFINITE;
}

void Person::WriteMicroRecord()
{
    // create output variables for linked actors
    long nSpouseID = -1; if (lSpouse) nSpouseID = lSpouse->actor_id;
    long nParentID = -1; if (lParent) nParentID = lParent->actor_id;

    // Push the fields into the output record.

    // ============================================================
    // When adding additional variables, this list has to be extended
    // ============================================================
    out_csv << actor_id;            // Actor ID
    out_csv << actor_weight;        // Weight
    out_csv << time;                // Time
    out_csv << nSpouseID;           // Spouse ID
    out_csv << nParentID;           // Parent ID
    out_csv << time_of_birth;       // Time of birth
    out_csv << (int)sex;            // Sex

    // 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;
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Pre- and Post-Simulation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void PreSimulation()
{
    // In the pre-simulation phase of MicroDataOutput.mpp module, the micro-data file is prepared
    // for writing records in the simulation. If the user selects micro-data output, a file is
    // opened and the header row is written containing all selected states

    if (WriteMicrodata)
    {
        // ============================================================
        // When adding additional variables, this list has to be extended
        // ============================================================
        std_string myString = "ID,";        // Actor ID
        myString = myString + "WEIGHT,";    // Weight
        myString = myString + "TIME,";      // Time
        myString = myString + "SPOUSE_ID,"; // Spouse
        myString = myString + "PARENT_ID,"; // Parent
        myString = myString + "BIRTH,";     // Time of birth
        myString = myString + "MALE,";      // Sex

        out_csv.open(MicroRecordFileName);
                out_csv.write_header(myString);
        }
}

void PostSimulation()
{
    if (WriteMicrodata) { out_csv.close(); } // closing the file
}
```

Micro Data Output: TablesPopulation.mpp

This module contains the output tables of the model

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Population Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_PopulationTables             //EN Population
{
    TotalPopulation,
    StartingPopulationSummary
};

table Person TotalPopulation                //EN Total Population
{
    {
        unit,                               //EN Persons at the beginning of each year
        duration()                          //EN Average population (total time lived) in year
    }
    * calendar_year
    * sex +
};
```

```
table Person StartingPopulationSummary      //EN Starting Population Summary
[person_type == PT_START]
{
    {
        unit                                //EN Persons
    }
    * year_of_birth +
    * sex +
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Birth Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_BirthTables                  //EN Births
{
    Births
};

table Person Births                         //EN Births
{
    {
        transitions(is_alive, FALSE, TRUE)  //EN Births
    }
    *calendar_year
    * sex +
};
```

Step 2: Birthdays

Model Description

At this step we add a module for birthday events to the model template. Unlike calendar year changes which happen to all actors at the same time and are thus implemented by a common clock, birthdays happen at individual pints in time throughout the year and are therefore implemented on the individual level. Besides the maintenance of a state for age in full years (which could be implemented directly as a self-scheduling state), the explicit modeling of a birthday event prepares the model for routines which typically happen on birthdays, e.g. the update of longitudinal accounts. All new code added at this step is contained in the new module BirthDays.mpp.

The new module: BirthDays.mpp

This module implements birthday events of the person actors. The module showcases the implementation of a clock event: a state integer_age is updated at yearly intervals. The module also demonstrates the typical arrangement of code within a regular simulation module:

- Introduction of new types: here an age range for the allowed life span. It is needed as a dimension of parameters (e.g. a lifetable) and as a table dimension.
- An “actor block” for the declaration of new states (here: integer_age and a variable to store the time of the next birthday) and new events: the birthday.
- The implementation of new functions and events. In our case the implementation of the birthday event. Like all events birthdays have two functions: a timing function returning the next occurrence of the event, and the event function performing all code to happen at the occurrence of the event.

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////////
//

range AGE_RANGE { 0, 105 };                                    //EN Age Range

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states and functions
////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
```

```
{
    AGE_RANGE   integer_age = { 0 };                        //EN Age

    TIME        time_next_birthday = { TIME_INFINITE };     //EN Time of next birthday
    event       timeBirthdayEvent, BirthdayEvent;           //EN Birthday Event
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Event Implementation
////////////////////////////////////////////////////////////////////////////////////////////
//

/* NOTE(Person.BirthdayEvent, EN)
   At each birthday integer age is incremented and all code to be performed at birthdays is
executed
*/

TIME Person::timeBirthdayEvent()
{
    if ( integer_age == 0 ) return time_of_birth + 1.0;
    else return time_next_birthday;
}

void Person::BirthdayEvent()
{
    // Increment integer age
    if (integer_age < MAX(AGE_RANGE)) integer_age++;

    // Code to be performed at each birthday can be entered here

    // Set clock for next birthday
    time_next_birthday = WAIT(1);
}
```

Step 3: Base Mortality

Model Description

At this step we add a module for mortality based on a period life table of projected mortality rates. This is the base version of Mortality resembling a typical macro population projection approach.

The new module: MortalityStandardLifeTable.mpp

This module implements a simple model of mortality. People can die at any moment of time based on age-specific period mortality rates. At death, the state is_alive is set to FALSE and the Modgen function Finish() is called which clears up memory space.

This module is a typical implementation of a stochastic event depending on hazard rates. The time function calculates a random waiting time to death based on the age-specific hazard rates of death. The time function is very typical for time functions scheduling an event in continuous time. If a person is at risk of the event, an exponentially distributed random waiting time is drawn and the event time returned. Whenever a state that influences the waiting time changes (in this case only integer_age), a new waiting time is drawn automatically based on the updated rate and the event is automatically re-scheduled.

The module is prepared for being combined with or replaced by a refined model accounting for more detailed personal characteristics than age and sex. For this purpose a logical state use_base_mortality is introduced and initialized with TRUE; Other modules can override the waiting time function of this module by setting the state to FALSE whenever an alternative model is applied.

```
////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////
//

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

parameter_group PG02_OverallMortality                              //EN Overall mortality
{
    MortalityTable
};
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states and functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    logical use_base_mortality = { TRUE };                      //EN Use the base version
    event       timeMortalityEvent, MortalityEvent;                             //EN Mortality
event
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation of functions and events
////////////////////////////////////////////////////////////////////////////////////////////////
//

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

    // check if a person is at risk
    if (dMortalityHazard > 0.0 && in_projected_time && ever_resident && use_base_mortality)
    {
        // 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(3)) / 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;
}

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

Step 4: Base Fertility

Model Description

At this step we add a module for fertility based on projected age-specific period fertility rates. This is the base version of fertility resembling a typical macro population projection approach.

The new module: FertilityAgePeriod.mpp

This module implements fertility based on age-specific fertility rates. This module is a microsimulation implementation of a typical cohort component model based on published population projection data.

The model is prepared for being complemented or over-ridden by an alternative refined fertility model. This is done by three logical states:

- The state use_base_fertility_model is initialized as TRUE; it indicates that the base model is to be used. When adding another model choice this flag can be changed in another module.
- The state use_base_fertility_for_alignment is initialized with FALSE; it indicates if another model is to be aligned to the fertility outcome of this base model.
- The state baby_looking_for_mother is set to TRUE at a birth event if the base model is used for alignment only and the actual birth has to be assigned to another person of the population.

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////////
//

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

////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN Age distribution of fertility
    double  AgeSpecificFertility[FERTILE_AGE_RANGE][SIM_YEAR_RANGE];

    //EN Sex ratio (male per 100 female)
```

```
    double  SexRatio[SIM_YEAR_RANGE];
};

parameter_group PG03a_Fertility_Model_A  //EN Fertility Base Model
{
    AgeSpecificFertility, SexRatio
};


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor declarations
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    logical use_base_fertility_model = { TRUE };            //EN Use the base model
    logical use_base_fertility_for_alignment = { FALSE };   //EN Use the model for alignment
    logical baby_looking_for_mother = { FALSE };            //EN A birth is still to be created

    //EN Indicator that perion is a potential mother
    logical is_potential_mother = (sex == FEMALE && WITHIN(FERTILE_AGE_RANGE, integer_age)
    && in_projected_time && ever_resident) ? TRUE : FALSE;

    int     parity;                                         //EN Parity
    event       timeBirthEvent, BirthEvent;                     //EN Birth event
};


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Event Implementations
////////////////////////////////////////////////////////////////////////////////////////////////
//

TIME Person::timeBirthEvent()
{
    double dEventTime = TIME_INFINITE;
    double dHazard = 0.0;

    if (is_potential_mother && (use_base_fertility_model || use_base_fertility_for_alignment))
    {
        dHazard = AgeSpecificFertility[RANGE_POS(FERTILE_AGE_RANGE, integer_age)]
                [RANGE_POS(SIM_YEAR_RANGE, calendar_year)];
        if (dHazard > 0.0) dEventTime = WAIT(-TIME(log(RandUniform(3)) / dHazard));
    }
    return dEventTime;
}

void Person::BirthEvent()
{
    // event applies to individual without alignment
    if (use_base_fertility_model)
    {
        parity++;                   // increment parity
        auto peChild = new Person;  // Create and point to a new actor
        peChild->Start(NULL, this);   // Call Start() function of baby and pass own address
    }
    else if (use_base_fertility_for_alignment)
    {
        baby_looking_for_mother = TRUE;
    }
}
```

Update of the Start() function in PersonCore.mpp

The Start() function is updated in order to initialize all states at the birth of a baby born in the simulation. In this case values do not come from the starting popiulation file but have to be derived otherwise, e.g. by accessing mother's characteristics (e.g. for setting the time) or by sampling (e.g. sex according to a parameter for sex ratio).

```
    else if (person_type == PT_CHILD) // Person born in simulation
    {
        // (A) States corresponding to Starting population file variables
        if (RandUniform(4) < 100.0 / (100.0 + SexRatio[RANGE_POS(SIM_YEAR_RANGE, calendar_year)]))
        {
            sex = FEMALE;
        }
        else sex = MALE;
        time = pePers->time;
        family_role = FR_CHILD;

        // (B) Other states
        time_of_birth = time;
        calendar_year = ( int )time_of_birth;

        // (C) Links to head resp. spouse
        peHHead = pePers;
    }
    //
```

Step 5: Education Fate Base Model

Model Description

At this step we add the base “fate” module for school outcome. Based on the information in the starting population and parameters by year of birth and sex, the final school outcome is decided at birth.

The new EducBaseFate.mpp Module

This module sets the base education fate and stores the underlying individual probabilities of education progressions. The education fate is decided at birth. There are three different outcomes: low, medium, high. The module sets three states at birth:

- educ_fate: the individual outcome in 3 levels low, medium, high
- educ_prob1: the individual probability to move from low to medium
- educ_prob2: the individual probability to move from medium to high

Probabilities to progress from low to medium and from medium to high are given by two parameters by year of birth and sex. The way how the education fate is decided depends on the year of birth and the person type.

- Persons from the starting population: the education outcome is taken from the starting population file if the person was born before the years of birth covered by the parameters. This means, up to two age cut-offs, the information of the starting population is ignored at all, or only used to decide the first progression (low to medium, but not medium to high which is modeled).
- Persons born in the simulation: the education fate is decided based on the parameters.

The function SetEducBaseFate() which decides the fate is hooked to the SetAlive() Event of the PersonCore.mpp module, ie. is called directly at birth after the actor enters the simulation and all actor sets are available and family links are operational.

This module is the base version for the education fate. As the probabilities on which the decision was based are stored, the outcome can be adjusted by a refined model in order to account for additional individual-level characeristics (like parents’ education) and/or to aligne the aggregated outcome to target rates or constraints.

When adding this module, some adaptations are necessary in other modules:

- The state educ_startpop has to be initialized in the Start() function in the PersonCore.mpp module.
- The year of birth ranges for the model parameters depend on the country and time context. The according ranges YOB_EDUC_PROG1 and YOB_EDUC_PROG2 are accordingly added in _CountryContext.mpp

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////////
//

classification EDUC_LEVEL3              //EN Education level
{
    EL3_LOW,                            //EN Low
    EL3_MEDIUM,                         //EN Medium
    EL3_HIGH                            //EN High
};

classification EDUC_LEVEL5              //EN Education Level
{
    EL5_LOW,                            //EN Low
    EL5_MEDIUMV,                        //EN Medium - vocational
    EL5_MEDIUMG,                        //EN Medium - general
    EL5_HIGHV,                          //EN High - vocational
    EL5_HIGHG                           //EN High - general
};

aggregation EDUC_LEVEL3, EDUC_LEVEL5    //EN Aggregation of Education Levels
{
    EL3_LOW,    EL5_LOW,
    EL3_MEDIUM, EL5_MEDIUMV,
    EL3_MEDIUM, EL5_MEDIUMG,
    EL3_HIGH,   EL5_HIGHV,
    EL3_HIGH,   EL5_HIGHG
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN Education progression probability low -> medium
    double  EducProg1[SEX][YOB_EDUC_PROG1];

    //EN Education progression probability medium -> high
    double  EducProg2[SEX][YOB_EDUC_PROG2];
};

//EN Education Base Fate Model
parameter_group PG_EducBaseFate
{
    EducProg1,  EducProg2
};

////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
// Actor States and Events
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    EDUC_LEVEL3  educ_fate = { EL3_LOW };        //EN Primary Education Fate
    EDUC_LEVEL5  educ_startpop = { EL5_LOW };    //EN Primary Education Fate

    double educ_prob1 = { 0.0 };                 //EN Base prob. progressing low medium
    double educ_prob2 = { 0.0 };                 //EN Base prob. progressing medium high

    void SetEducBaseFate();                      //EN Setting the education fate
    hook SetEducBaseFate, SetAliveEvent;         //EN Hook SetEducBaseFate to SetAlive
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void Person::SetEducBaseFate()
{
    // Person born in simulation
    if ( person_type == PT_CHILD )
    {
        educ_prob1 = EducProg1[sex][RANGE_POS(YOB_EDUC_PROG1,year_of_birth)];
        educ_prob2 = EducProg2[sex][RANGE_POS(YOB_EDUC_PROG2,year_of_birth)];
        EDUC_LEVEL3 eolFate = EL3_LOW;
        if ( RandUniform(5) < educ_prob1 ) eolFate = EL3_MEDIUM;
        if ( eolFate == EL3_MEDIUM && RandUniform(6) < educ_prob2 )  eolFate = EL3_HIGH;
        educ_fate = eolFate;
    }
    // Person from starting population
    else if ( person_type == PT_START )
    {
        // Case 1: take the education from the starting population
        if ( year_of_birth < MIN(YOB_EDUC_PROG2))
        {
            educ_fate = EDUC_LEVEL5_To_EDUC_LEVEL3(educ_startpop);
        }
        // Case 2: take first progression from the starting population
        else if ( year_of_birth < MIN(YOB_EDUC_PROG1))
        {
            educ_prob2 = EducProg2[sex][RANGE_POS(YOB_EDUC_PROG2,year_of_birth)];
            EDUC_LEVEL3 eolFate = EL3_LOW;
            if ( educ_startpop != EL5_LOW ) eolFate = EL3_MEDIUM;
            if ( eolFate == EL3_MEDIUM && RandUniform(7) < educ_prob2 )  eolFate = EL3_HIGH;
            educ_fate = eolFate;
        }
        // Case 3: model from parameters ignoring starting population values
        else
        {
            educ_prob1 = EducProg1[sex][RANGE_POS(YOB_EDUC_PROG1,year_of_birth)];
            educ_prob2 = EducProg2[sex][RANGE_POS(YOB_EDUC_PROG2,year_of_birth)];
            EDUC_LEVEL3 eolFate = EL3_LOW;
            if ( RandUniform(8) < educ_prob1 ) eolFate = EL3_MEDIUM;
            if ( eolFate == EL3_MEDIUM && RandUniform(9) < educ_prob2 )  eolFate = EL3_HIGH;
            educ_fate = eolFate;
        }
    }
}

table Person TabEducTest //EN Educ Table
[is_alive]
```

```
{
    person_type+ *
    {
        unit
    }
    * year_of_birth
    * educ_fate+
};
```

Initializations in the Start() Function

The state educ_startpop is added to the list of variables initiated in the Start() Function.

```
    // Initialize states
    if (person_type == PT_START) // Person comes from starting population
    {
        // (A) States from Starting population file
        time                = peObservation->pmc[PMC_BIRTH] + RandUniform(2);
        sex                 = (SEX)(int)peObservation->pmc[PMC_SEX];
        family_role         = (FAM_ROLE)( int )peObservation->pmc[PMC_ROLE];
        educ_startpop       = (EDUC_LEVEL5)(int)peObservation->pmc[PMC_EDUC];
```

Context-Specific Ranges in _CountryContext.mpp

```
range YOB_EDUC_PROG1{ 1990, 2050 };             //EN Year of birth
range YOB_EDUC_PROG2{ 1980, 2050 };             //EN Year of birth
```

Step 6: Primary Education Fate Refined Model

Model Description

At this step we add the refined “fate” module for school outcome. It adjusts the overall probabilities from the base module to progress between levels by a set of relative factors (odds ratios) by individual characteristics. In the current implementation parent’s education is introduced as relative factor. The added module allows to easily expand the list of distinguished population groups. Users are also given a choice if and how the refined model is used. One option is calibrating the aggregate outcomes by sex and district to the base model for all years of birth. The second option calibrates the model just for one birth cohort from which onward the refined module is used. All simulated future trends are then driven entirely by composition effects.

The new EducRefinedFate.mpp Module

This module allows to add relative factors (odds ratios) to the modeling of education outcomes. Currently mother’s education by sex is introduced. The relative factors are used to modify the base probabilities of school progressions. Aggregated outcomes by sex are calibrated in order to match the base model’s outcome. This calibration is performed either (1) for all years of birth, or (2) just once. In the first case, aggregate outcomes by sex are identical, but the model picks different children entering and graduating school, accounting for the relative differences (odds ratios) by mother’s education. In the second case, the calibration is done for a selected year of birth. For all following cohorts, parameters are frozen and all trends result from composition effects due to the changing educational composition of mothers. The module only affects persons born in the simulation as mother’s education is unknown for others.

Parameters:

- Model selection: the 3 choices are (1) Use Base Model, (2) Use the refined model calibrated for all years of birth, (3) Use the refined model calibrated once.
- Odds for progressing from low to medium by parents education
- Odds for progressing from medium to high education by parents education
- The first birth cohort from which onwards the refined model is used, which can be any year beginning from the starting year of the simulation.

The module can be added or removed from the model without requiring modification in other modules. The only exception is the requirement to initialize parents' education in the Start() function in PersonCore.mpp.

In its core, this module consists of two functions.

- The function SetEduc1AdjustmentFactors() calculates the required adjustment factors (in the form of log odds by district and sex) to be applied in addition to the relative factors in order to get probabilities further broken down by mother's education. The calculations are performed once at the end of each relevant calendar year as at this point in time the educational composition of mothers of all born in this year is known. Adjustment factors are either calculated once or updated each year according to the user's model selection.
- The function AdjustEducOne() applies the adjustments at the Person level at the first end of the year after birth. Thereby this module modifies the three "fate" states introduced in the base model, namely the two individual progression probabilities educ_prob1 and educ_prob2, as well as random outcome educ_fate based on the new probabilities.

How to change or expand the list of relative factors

- The list of population groups is declared in the classification educ_group. The classification can be changed or expanded whereby a person can belong only to one of the groups. (For example, when adding an additional dimension ethnicity, all combinations of ethnicity and mother's education have to be listed)
- The individual group membership is a derived state educ_group. When changing the levels of educ_group, the calculation of the derived state has to be updated accordingly.
- No other code changes are required

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Sets
////////////////////////////////////////////////////////////////////////////////////////////
//

actor_set Person asSimBornAge0[sex][educ_group]
    filter is_alive && person_type == PT_CHILD && integer_age == 0;

////////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
classification EDUC_GROUP           //EN Population Groups
{
    EG_00,                          //EN Parents education low
    EG_01,                          //EN Parents education medium
    EG_02                           //EN Parents education high
};

classification EDUC_MODEL           //EN Education Model Selection
{
    EM_BASE,                        //EN Use base model
    EM_REFINED_ALIGNALL,            //EN Use refined model aligned to base for all birth cohorts
    EM_REFINED_ALIGNONCE            //EN Use refined model aligned to base once
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    double      EducProg1Odds[EDUC_GROUP][SEX];     //EN Odds of progressing low->medium
    double      EducProg2Odds[EDUC_GROUP][SEX];     //EN Odds of progressing medium->high
    EDUC_MODEL  EducModel;                          //EN Model Selection
    int         EducFirstCohortRefinedModel;        //EN First birth cohort to apply refined model
};

parameter_group PG_EducRefinedFate                  //EN Education Fate - Refined Model
{
    EducProg1Odds, EducProg2Odds,
    EducFirstCohortRefinedModel
};

parameter_group PG_SchoolFate                       //EN Education Fate
{
    EducModel,
    PG_EducBaseFate,
    PG_EducRefinedFate
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states and functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Globals                                       //EN Actor Globals
{
    double alignment_educ_medium[SEX];              //EN Education alignment low to medium
    double alignment_educ_high[SEX];                //EN Education alignment medium to high
};

actor Clock
{
    void    SetEducAdjustmentFactors();             //EN Function calculating calibration factors
    hook    SetEducAdjustmentFactors, EndYearClock; //EN Hook to clock yearend

    double  AdjustedProbability(double dProb, double dLogOddEduc, double dLogOddAdjust);
};

actor Person
{
    EDUC_LEVEL3  educ_parents = { EL3_LOW };            //EN Parents education
    logical      educ_parents_is_known = { FALSE };     //EN Parents education is known
```

```
    //EN Education risk group
    EDUC_GROUP   educ_group = (educ_parents == EL3_LOW) ? EG_00 :
    (educ_parents == EL3_MEDIUM) ? EG_01 : EG_02;

    void AdjustEduc(); hook AdjustEduc, YearEnd;

    void SetEducParents(); hook SetEducParents, SetAliveEvent;
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void Person::SetEducParents()
{
    EDUC_LEVEL3  cEducParents = EL3_LOW;
    if (lParent)
    {
        cEducParents = lParent->educ_fate;
        educ_parents_is_known = TRUE;

        if (lParent->lSpouse && lParent->lSpouse->educ_fate > lParent->educ_fate)
        {
            cEducParents = lParent->lSpouse->educ_fate;
        }
    }
    educ_parents = cEducParents;
}

void Clock::SetEducAdjustmentFactors()
{
    if (clock_year >= MIN(SIM_YEAR_RANGE) && (
        (EducModel == EM_REFINED_ALIGNALL && clock_year >= EducFirstCohortRefinedModel) ||
        (EducModel == EM_REFINED_ALIGNONCE && clock_year == EducFirstCohortRefinedModel)))
    {
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            // calculate total population for given sex
            double nTotalPop = 0.0;
            for (int dGroup = 0; dGroup < SIZE(EDUC_GROUP); dGroup++)
            {
                nTotalPop = nTotalPop + asSimBornAge0[nSex][dGroup]->Count();
            }

            // Run Adjustment for school entry
            double dFactorEntry = 0.0;
            if (nTotalPop > 0.0)
            {
                int nIterations = 0;
                double dResultProb = 10.0;
                double dTargetProb = EducProg1[nSex][RANGE_POS(YOB_EDUC_PROG1, clock_year)];
                double dLower = -10.0;
                double dUpper = 10.0;
                double dCenter = 0.0;
                while (abs(dResultProb - dTargetProb) > 0.0001 && nIterations < 10000)
                {
                    nIterations++;
                    dCenter = (dLower + dUpper) / 2.0;
                    dResultProb = 0.0;
                    for (int nGroup = 0; nGroup < SIZE(EDUC_GROUP); nGroup++)
                    {
                        dResultProb = dResultProb + (asSimBornAge0[nSex][nGroup]->Count() /
nTotalPop) *
```

```
                            AdjustedProbability(dTargetProb, log(EducProg1Odds[nGroup][nSex]),
dCenter);
                    }
                    if (dTargetProb > dResultProb) dLower = dCenter;
                    else dUpper = dCenter;
                }
                dFactorEntry = dCenter;
            }
            // set factor
            asGlobals->Item(0)->alignment_educ_medium[nSex] = dFactorEntry;

            // Run Adjustment for medium high progression
            double dFactorGrad = 0.0;
            if (nTotalPop > 0.0)
            {
                int nIterations = 0;
                double dResultProb = 10.0;
                double dTargetProb = EducProg2[nSex][RANGE_POS(YOB_EDUC_PROG2, clock_year)];
                double dLower = -10.0;
                double dUpper = 10.0;
                double dCenter = 0.0;
                while (abs(dResultProb - dTargetProb) > 0.0001 && nIterations < 10000)
                {
                    nIterations++;
                    dCenter = (dLower + dUpper) / 2.0;
                    dResultProb = 0.0;
                    for (int nGroup = 0; nGroup < SIZE(EDUC_GROUP); nGroup++)
                    {
                        dResultProb = dResultProb + (asSimBornAge0[nSex][nGroup]->Count() /
nTotalPop) *
                            AdjustedProbability(dTargetProb, log(EducProg2Odds[nGroup][nSex]),
dCenter);
                    }
                    if (dTargetProb > dResultProb) dLower = dCenter;
                    else dUpper = dCenter;
                }
                dFactorGrad = dCenter;
            }
            // set factor
            asGlobals->Item(0)->alignment_educ_high[nSex] = dFactorGrad;
        }
    }
}

double Clock::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);
}

void Person::AdjustEduc()
{
    // Adjustment for selection of  refined model aligned to base for all birth cohorts
    if (person_type == PT_CHILD && integer_age == 0 && calendar_year >= MIN(SIM_YEAR_RANGE) &&
        calendar_year >= EducFirstCohortRefinedModel && EducModel == EM_REFINED_ALIGNALL)
    {
        educ_prob1 = lClock->AdjustedProbability(
            educ_prob1,
            log(EducProg1Odds[educ_group][sex]),
            asGlobals->Item(0)->alignment_educ_medium[sex]);
        educ_prob2 = lClock->AdjustedProbability(
            educ_prob2,
            log(EducProg2Odds[educ_group][sex]),
            asGlobals->Item(0)->alignment_educ_high[sex]);
        EDUC_LEVEL3 eolFate = EL3_LOW;
```

```
        if (RandUniform(10) < educ_prob1) eolFate = EL3_MEDIUM;
        if (eolFate == EL3_MEDIUM && RandUniform(11) < educ_prob2)  eolFate = EL3_HIGH;
        educ_fate = eolFate;
    }
    else if (person_type == PT_CHILD && integer_age == 0 && calendar_year >= MIN(SIM_YEAR_RANGE)
&&
        calendar_year >= EducFirstCohortRefinedModel && EducModel == EM_REFINED_ALIGNONCE)
    {
        educ_prob1 = lClock->AdjustedProbability(
            EducProg1[sex][RANGE_POS(YOB_EDUC_PROG1, EducFirstCohortRefinedModel)],
            log(EducProg1Odds[educ_group][sex]),
            asGlobals->Item(0)->alignment_educ_medium[sex]);
        educ_prob2 = lClock->AdjustedProbability(
            EducProg2[sex][RANGE_POS(YOB_EDUC_PROG2, EducFirstCohortRefinedModel)],
            log(EducProg2Odds[educ_group][sex]),
            asGlobals->Item(0)->alignment_educ_high[sex]);
        EDUC_LEVEL3 eolFate = EL3_LOW;
        if (RandUniform(12) < educ_prob1) eolFate = EL3_MEDIUM;
        if (eolFate == EL3_MEDIUM && RandUniform(13) < educ_prob2)  eolFate = EL3_HIGH;
        educ_fate = eolFate;
    }
}


table Person PareducTabTest //EN Parents education
{ {
        value_out(educ_parents_is_known) / unit

}
*year_of_birth
};
```

Step 7: Refined Mortality

Model Description

At this step we add the refined mortality module accounting for life expectancy differentials by educational attainment. The module is parameterized by period life expectancies at age 30 and age 65. The module automatically calibrates the period mortality rates of the base module in order to meet the target parameters of life expectancy. Additionally, the outcome can be 'aligned back' to the original overall mortality rates keeping the relative differences in risks between education groups.

The new MortalityByEducation.mpp Module

This module implements differential mortality by population "mortality groups". While the grouping is generic, in the current implementation it refers to educational attainment, but the list can easily be extended or changed. The module is optional, and the user can choose between various options:

- Disable the module: the overall mortality rates of the base model apply and this module is switched off
- Model without alignment: The base rates are adjusted to meet life expectancy parameters by population group. Parameters are the remaining period life expectancy at age 30 and age 65 by sex. This adjustment is made in the pre-simulation phase where new parameter tables are generated.
- Model with alignment: after the initial calibration of rates to meet life expectancy targets by group, the resuling rates are proportionally scaled back in order to keep the overall mortality rates of the base model while accounting for the relative mortality differences by population group. This alignment is performed in yearly intervals during the simulation accounting for the changing population composition by risk group.

How to change or add the population groups for relative risks?

The list of population groups for which relative risks are applied is generic and can be modified and extended easily.

- The classification CHILD_MORTALITY_GROUP lists all population groups distinguished.
- The state mortality_group contains the individual group membership and has to be adapted if the list of categories is modified.

Besides these two changes no coding is required. The calibration routines automatically adjust the hazards according to the population composition by group membership.

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////////
//

classification SELECTED_MORTALITY_MODEL         //EN Mortality Model Options
{
    SMM_MACRO,                                  //EN Disable child mortality model
    SMM_MICRO_NOT_ALIGNED,                      //EN Child mortality model without alignment
    SMM_MICRO_ALIGNED                           //EN Aligned to overall mortality rates
};

classification MORTALITY_GROUP                  //EN Population group for mortality
{
    MG_00,                                      //EN Low education
    MG_01,                                      //EN Medium education
    MG_02                                       //EN High education
};

classification LIFE_EXPECT                      //EN Life Expectancy Parameters
{
    LE_30,                                      //EN Life expectancy at 30
    LE_65                                       //EN Life expectancy at 65
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN Child mortality model selection
    SELECTED_MORTALITY_MODEL  SelectedMortalityModel;

    //EN Period life expectancy
    double  LifeExpectancy[SEX][LIFE_EXPECT][SIM_YEAR_RANGE][MORTALITY_GROUP];

    //EN Mortality Trend
    model_generated double MortalityByGroup[SEX][MORTALITY_GROUP][AGE_RANGE][SIM_YEAR_RANGE];
};

parameter_group PG02_RefinedMortality               //EN Mortality by Education
{
    LifeExpectancy
};

parameter_group PG_Mortality                    //EN Mortality
{
    SelectedMortalityModel,
    PG02_OverallMortality,
    PG02_RefinedMortality

};

////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
// Actor Globals
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Globals
{
    void    MortalityCalibration();           //EN Mortality calibration
    double  CurrentAlignedMortality[MORTALITY_GROUP][AGE_RANGE][SEX];
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Clock
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Clock
{
    void CallMortalityCalibration();        //EN Call mortality calibration event
    hook CallMortalityCalibration, StartYearClock;
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Person
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    //EN Child mortality risk group
    MORTALITY_GROUP mortality_group = (educ_fate == EL3_LOW) ? MG_00 :
        (educ_fate == EL3_MEDIUM) ? MG_01 : MG_02;

    // Current aligned mortality
    double current_aligned_mortality = { 0.0 };

    //EN Activates Module at birth if selected
    void ActivateRefinedMortalityModule(); hook ActivateRefinedMortalityModule, Start;

    event timeRefinedMortalityEvent, RefinedMortalityEvent; //EN Mortality Event
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Person functions implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void Person::ActivateRefinedMortalityModule()
{
    if ( SelectedMortalityModel != SMM_MACRO)  use_base_mortality = FALSE;
}

TIME Person::timeRefinedMortalityEvent()
{
    double dEventTime = TIME_INFINITE;

    if ( !use_base_mortality && in_projected_time && ever_resident)
    {
        double dHazard = MortalityByGroup[sex][mortality_group][integer_age]
            [RANGE_POS(SIM_YEAR_RANGE, calendar_year)];

        // aligned model
        if (SelectedMortalityModel == SMM_MICRO_ALIGNED)
        {
```

```
            dHazard = current_aligned_mortality;
        }
        // if hazard is positive calculate event time
        if (dHazard > 0) dEventTime = WAIT(-log(RandUniform(15)) / dHazard);
    }
    // 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 if (!use_base_mortality) return time_of_birth + MAX(AGE_RANGE) + 0.9999;
    else return TIME_INFINITE;
}

void Person::RefinedMortalityEvent()
{
    MortalityEvent();
}

////////////////////////////////////////////////////////////////////////////////////////////
//
// Clock functions implementation
////////////////////////////////////////////////////////////////////////////////////////////
//

void Clock::CallMortalityCalibration()
{
    if (clock_year >= MIN(SIM_YEAR_RANGE) && SelectedMortalityModel == SMM_MICRO_ALIGNED)
    {
        asGlobals->Item(0)->MortalityCalibration();
    }
}

////////////////////////////////////////////////////////////////////////////////////////////
//
// Globals functions implementation
////////////////////////////////////////////////////////////////////////////////////////////
//

void Globals::MortalityCalibration()
{
    // local matrices
    double  dDeaths[SIZE(AGE_RANGE)][SIZE(SEX)];                        // Number expected deaths
    double  dProb[SIZE(AGE_RANGE)][SIZE(SEX)];                          // Death probability
    long    nPop[SIZE(AGE_RANGE)][SIZE(SEX)][SIZE(MORTALITY_GROUP)];    // Pop sizes
    double  dCalibrator[SIZE(AGE_RANGE)][SIZE(SEX)];                    // Baseline Hazard in
simulation
    int     nYear = asClock->Item(0)->clock_year - MIN(SIM_YEAR_RANGE); // Current Yyear since
start

    // 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(AGE_RANGE); nAge++)
    {
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            dProb[nAge][nSex] = 1.0 - exp(-MortalityTable[nSex][nAge][nYear]);
            dDeaths[nAge][nSex] = 0.0;
            for (int nGroup = 0; nGroup < SIZE(MORTALITY_GROUP); nGroup++)
            {
                nPop[nAge][nSex][nGroup] = 0.0;
            }
        }
    }

    // Population Count
    // calculates population sizes nPop by age, sex, risk group
```

```
    // calculates expected deaths dDeaths by age and sex
    for (long nJ = 0; nJ < asAllPersons->Count(); nJ++)
    {
        auto prPerson = asAllPersons->Item(nJ); //NOTE: to be changed to residents if migration is
modeled
        dDeaths[prPerson->integer_age][prPerson->sex] = dDeaths[prPerson->integer_age][prPerson-
>sex]
            + dProb[prPerson->integer_age][prPerson->sex];
        nPop[prPerson->integer_age][prPerson->sex][prPerson->mortality_group]++;
    }

    // find calibrated baselines
    for (int nAge = 0; nAge < SIZE(AGE_RANGE); 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--;
                dCalibrator[nAge][nSex] = (dLower + dUpper) / 2.0;

                dNumberDeaths = 0.0;

                //Calculate numer of deaths for given dCalibrator
                for (int nGroup = 0; nGroup < SIZE(MORTALITY_GROUP); nGroup++)
                {
                    dNumberDeaths = dNumberDeaths + nPop[nAge][nSex][nGroup]
                        * (1 - exp(-dCalibrator[nAge][nSex] *
MortalityByGroup[nSex][nGroup][nAge][nYear]));
                }
                // shrink search interval
                if (dNumberDeaths > dDeaths[nAge][nSex]) dUpper = dCalibrator[nAge][nSex];
                else dLower = dCalibrator[nAge][nSex];
            }
        }
    }

    // set global mortality by group, age and sex
    for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
    {
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            for (int nGroup = 0; nGroup < SIZE(MORTALITY_GROUP); nGroup++)
            {
                CurrentAlignedMortality[nGroup][nAge][nSex]
                    = MortalityByGroup[nSex][nGroup][nAge][nYear] * dCalibrator[nAge][nSex];
            }
        }
    }

    for (long nJ = 0; nJ < asAllPersons->Count(); nJ++)
    {
        auto prPerson = asAllPersons->Item(nJ); //NOTE: to be changed to residents if migration is
modeled
        prPerson->current_aligned_mortality
            = CurrentAlignedMortality[prPerson->mortality_group][prPerson->integer_age][prPerson-
>sex];
    }
```

```
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Pre-Simulation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void PreSimulation()
{
    // In the pre-simulation phase mortality rates by population group are found by binary search.
    // For each simulated year, the overall mortality table is adjusted to obtain target life
    // expectancies at age 65 and at age 30 for each distinguished population group.

    // Local variables
    double  dTarget, dCenter, dResult, dAlive, dDeaths, dLower, dUpper;
    int     nIterations;

    if (SelectedMortalityModel != SMM_MACRO)
    {
        // Find trend factors to fit the life expectancy at 65
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
            {
                for (int nGroup = 0; nGroup < SIZE(MORTALITY_GROUP); nGroup++)
                {
                    dTarget = LifeExpectancy[nSex][LE_65][nYear][nGroup];   // Target life
expectancy
                    dResult = 0.0;                                          // Search result: life
expectancy
                    dLower = 0.1;                                           // Lower limit of
calibration factor
                    dUpper = 10.0;                                          // Upper limit of
calibration factor
                    nIterations = 100000;                                   // Maximal iterations

                    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 = 65; nAge < SIZE(AGE_RANGE); nAge++)
                        {
                            // proportion of deaths in year: survival = exp(-hazard)
                            dDeaths = dAlive * (1 - exp(-MortalityTable[nSex][nAge][nYear] *
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;
                    }
                    // applying the best solution to the mortality parameter
                    for (int nAge = 65; nAge < SIZE(AGE_RANGE); nAge++)
                    {
                        MortalityByGroup[nSex][nGroup][nAge][nYear] = dCenter *
MortalityTable[nSex][nAge][nYear];
                    }
```

```
                }
            }
        }

        // Find trend factors to fit the life expectancy at 30 (while keeping the trend for 65+)
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
            {
                for (int nGroup = 0; nGroup < SIZE(MORTALITY_GROUP); nGroup++)
                {
                    dTarget = LifeExpectancy[nSex][LE_30][nYear][nGroup];   // Target life
expectancy
                    dResult = 0.0;                                          // Search result: life
expectancy
                    dLower = 0.1;                                           // Lower limit of
calibration factor
                    dUpper = 10.0;                                          // Upper limit of
calibration factor
                    nIterations = 100000;                                   // Maximal iterations

                    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 = 30; nAge < SIZE(AGE_RANGE); nAge++)
                        {
                            // proportion of deaths in year: survival = exp(-hazard)
                            if (nAge < 65) // Apply searched adjustment factor only for ages below
65
                            {
                                dDeaths = dAlive * (1 - exp(-MortalityTable[nSex][nAge][nYear] *
dCenter));
                            }
                            else // apply known factor for ages 65+
                            {
                                dDeaths = dAlive * (1 - exp(-
MortalityByGroup[nSex][nGroup][nAge][nYear]));
                            }
                            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;
                    }
                    // applying the best solution to the mortality parameter
                    for (int nAge = 30; nAge < 65; nAge++)
                    {
                        MortalityByGroup[nSex][nGroup][nAge][nYear] = dCenter *
MortalityTable[nSex][nAge][nYear];
                    }
                }
            }
        }

        // Copy over parameters for age < 30
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
```

```
        {
            for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
            {
                for (int nAge = 0; nAge < 30; nAge++)
                {
                    for (int nGroup = 0; nGroup < SIZE(MORTALITY_GROUP); nGroup++)
                    {
                        MortalityByGroup[nSex][nGroup][nAge][nYear] =
MortalityTable[nSex][nAge][nYear];
                    }
                }
            }
        }
    }
}

table Person CalibratedMortality                //EN CalibratedMortality
{
    sex + *
{
    max_value_out(current_aligned_mortality)
}
*integer_age
*calendar_year
};
```

Step 8: Refined Fertility

Model Description

At this step we add a refined fertility module able to account for different levels of childlessness and the different age patterns of first birth by education group.

The new FertilityByEducation.mpp Module

This module - in its current state - is a very simple extension of the base fertility model allowing to control for educational differences in first births, both by timing and final outcome, i.e. it allows to model childlessness by education, which is an important variable for the use of the model. Users can select to use or not to use the model by a selection parameter. Aggregate outcomes (births by age) are identical as in the base model. Besides the on/off switch, the model has two parameters, namely

- Cohort first birth rates by education for birth cohorts in reproductive age
- Childlessness by education for older birth cohorts

If switched on, the algorithm is as follows:

- The base module produces birth events, but does not assign the babies to a mother yet. The baby just knows the age of the mother.
- First birth events of this module flag women expecting a first birth
- The babies are first given to women of the appropriate age flagged for first birth
- The remaining babies are distributed randomly to women of the according age who are already mothers

The module initialized the state is_mother at the start of the simulation using the information from links in the starting population file as well as the parameters which allow to calculated the expected rates of childlessness by year of birth, education and age. The state is_mother is imputed by the function ImputeIsMother() in the following way:

- Using family links from the starting population file, all women living with children in the family are marked as mothers
- The number of expected mothers by age and education is calculated from the population sizes in the starting population and the first birth and childlessness parameters.

- If the number of expected mothers is higher than the number of recorded mothers (which should always be the case, as children can have moved out) childless women are selected randomly (for given age and education) and their status set to mother.

This module can be added or removed from the model without any code change in other modules. The only exception are the country/context specific time ranges set in _CountryContext.mpp.

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Sets
////////////////////////////////////////////////////////////////////////////////////////////
//

//EN Potential mothers for first birth by age
actor_set Person asWomenWaitingFirstBirth[fertile_age]
    filter is_alive && is_potential_mother && waits_for_first_baby;

//EN Potential mothers for higher order births by age
actor_set Person asPotentialMothersHigherBirths[fertile_age]
    filter is_potential_mother && is_mother;

//EN Childless women by year of birt and education
actor_set Person asChildlessWomen[year_of_birth][birth1_group] filter !is_mother && sex==FEMALE;



////////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////////
//

classification BIRTH1_GROUP                         //EN Population Groups
{
    B1G_00,                                         //EN Education Low
    B1G_01,                                         //EN Education Medium
    B1G_02                                          //EN Education High
};

classification SELECTED_FERTILITY_MODEL             //EN Fertility model options
{
    SFM_MACRO,              //EN Macro model (age only)
    SFM_ALIGNE_AGE          //EN Micro model with aligned number of births by age
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN Fertility model selection
    SELECTED_FERTILITY_MODEL selected_fertility_model;

    //EN First Birth Rates
```

```
    double  FirstBirthCohortRates[BIRTH1_GROUP][FERTILE_AGE_RANGE][YOB_BIRTH1];

    //EN Childlessnesss in older population
    double ChildlessnessYob[YOB_START50][BIRTH1_GROUP];
};

parameter_group PG03_Fertility_Top              //EN Fertility
{
    selected_fertility_model,
    PG03a_Fertility_Model_A,
    PG03b_Fertility_Model_B
};

parameter_group PG03b_Fertility_Model_B         //EN Fertility: Refined Version
{
    FirstBirthCohortRates
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor States and Functions
////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    //EN Population group for modeling of births
    BIRTH1_GROUP birth1_group = (educ_fate == EL3_LOW) ? B1G_00 :
        (educ_fate == EL3_MEDIUM) ? B1G_01 : B1G_02;

    FERTILE_AGE_RANGE   fertile_age = COERCE(FERTILE_AGE_RANGE, integer_age);   //EN Age

    logical is_mother = { FALSE };                          //EN Is Mother
    logical waits_for_first_baby = { FALSE };              //EN Waiting for first birth

    void SetFertilityModelSwitches();                       //EN Initializing model choices
    hook SetFertilityModelSwitches, Start;                  //EN hooked to Start function

    void MakeBaby();                                        //EN Function creating a baby
    event timeRefineModelBirthEvent, RefineModelBirthEvent; //EN Birth event
    event timeFirstBirthFlagEvent, FirstBirthFlagEvent;     //EN First birth flag event

    int linked_kids = count(mlChild);
};

actor Clock
{
    void ImputeIsMother();
    hook ImputeIsMother, StartYearClock;
};

////////////////////////////////////////////////////////////////////////////////////////////
////
//// Implementation
////////////////////////////////////////////////////////////////////////////////////////////
////

void Person::SetFertilityModelSwitches()
{
    if (selected_fertility_model == SFM_MACRO) // Macro model (age only)
    {
        use_base_fertility_model =  TRUE ;            // Use the base model
        use_base_fertility_for_alignment =  FALSE ;   // Base model not used for alignment
    }
    else                                              // Refined model with aligned births
    {
```

```
        use_base_fertility_model =  FALSE ;           // Do not use the base model
        use_base_fertility_for_alignment = TRUE ;     // Base model used for alignment
    }
}

TIME Person::timeFirstBirthFlagEvent()
{
    TIME dEventTime = { TIME_INFINITE };
    double dHazard = { 0.0 };
    if (is_potential_mother && !use_base_fertility_model && use_base_fertility_for_alignment
        && !is_mother && !waits_for_first_baby)
    {
        dHazard = FirstBirthCohortRates[birth1_group][RANGE_POS(FERTILE_AGE_RANGE,
fertile_age)][RANGE_POS(YOB_BIRTH1, year_of_birth)];
        if (dHazard > 0.0 ) dEventTime = WAIT(-log(RandUniform(16)) / dHazard);
    }
    return dEventTime;
}

void Person::FirstBirthFlagEvent()
{
    waits_for_first_baby = TRUE;
}

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

TIME Person::timeRefineModelBirthEvent()
{
    double dEventTime = TIME_INFINITE;
    if (baby_looking_for_mother) //EN The base model to which the model is aligned produced a baby
    {
        dEventTime = WAIT(0);
    }
    return dEventTime;
}

void Person::RefineModelBirthEvent()
{
    baby_looking_for_mother = FALSE;

    if (asWomenWaitingFirstBirth[RANGE_POS(FERTILE_AGE_RANGE,integer_age)]->Count() > 0)
    {
        auto peMother = asWomenWaitingFirstBirth[RANGE_POS(FERTILE_AGE_RANGE, integer_age)]-
>GetRandom(RandUniform(17));
        peMother->MakeBaby();
        peMother->is_mother = TRUE;
        peMother->waits_for_first_baby = FALSE;
    }
    else if ( asPotentialMothersHigherBirths[RANGE_POS(FERTILE_AGE_RANGE, integer_age)]->Count() >
0)
    {
        auto peMother = asPotentialMothersHigherBirths[RANGE_POS(FERTILE_AGE_RANGE, integer_age)]-
>GetRandom(RandUniform(18));
        peMother->MakeBaby();
    }
    else
    {
        //this should not happen
        //get the baby herself
        MakeBaby();
        is_mother = TRUE;
        waits_for_first_baby = FALSE;
```

```
    }
}


void Clock::ImputeIsMother()
{
    if (clock_year == MIN(SIM_YEAR_RANGE))
    {
        int nPopSize;
        int nPersons[SIZE(YOB_START15)][SIZE(BIRTH1_GROUP)];
        int nExpected[SIZE(YOB_START15)][SIZE(BIRTH1_GROUP)];
        int nRecorded[SIZE(YOB_START15)][SIZE(BIRTH1_GROUP)];
        // initialize with 0
        for (int nJ = 0; nJ < SIZE(YOB_START15); nJ++)
        {
            for (int nGroup = 0; nGroup < SIZE(BIRTH1_GROUP); nGroup++)
            {
                nPersons[nJ][nGroup] = 0; nExpected[nJ][nGroup] = 0; nRecorded[nJ][nGroup] = 0;
            }
        }
        // assign status is_mother where there are children in hh
        // count women by age and known mothers by age
        nPopSize = asAllPersons->Count();
        for (int nI = 0; nI < nPopSize; nI++)
        {
            auto prPerson = asAllPersons->Item(nI);
            if (WITHIN(YOB_START15, prPerson->year_of_birth) && prPerson->sex == FEMALE)
            {
                int nYOB = prPerson->year_of_birth - MIN(YOB_START15);
                nPersons[nYOB][prPerson->birth1_group] = nPersons[nYOB][prPerson->birth1_group] +
1;

                if (prPerson->linked_kids > 0 || (prPerson->lSpouse && prPerson->lSpouse-
>linked_kids >0))
                {
                    nRecorded[nYOB][prPerson->birth1_group] = nRecorded[nYOB][prPerson-
>birth1_group] + 1;
                    prPerson->is_mother = TRUE;
                }
            }
        }
        //calculate expected number of mothers by yob
        for (int nGroup = 0; nGroup < SIZE(BIRTH1_GROUP); nGroup++)
        {
            for (int nJ = 0; nJ < SIZE(YOB_START15); nJ++)
            {
                if (nJ < SIZE(YOB_START50)) // older population
                {
                    nExpected[nJ][nGroup] = round(nPersons[nJ][nGroup] * (1 -
ChildlessnessYob[nJ][nGroup]));
                }
                else // still in childbearing age
                {
                    double dChildless = 1.0;
                    for (int nAgeIndex = 0; nAgeIndex < SIZE(FERTILE_AGE_RANGE) && nJ +
MIN(YOB_START15) + MIN(FERTILE_AGE_RANGE) + nAgeIndex < MIN(SIM_YEAR_RANGE); nAgeIndex++)
                    {
                        dChildless = dChildless * exp(-FirstBirthCohortRates[nGroup][nAgeIndex][nJ
- SIZE(YOB_START50)]);
                    }
                    nExpected[nJ][nGroup] = round(nPersons[nJ][nGroup] * (1 - dChildless));
                }
            }
        }
        //randomly choose childless women to become mothers in order to meet target numbers
        for (int nGroup = 0; nGroup < SIZE(BIRTH1_GROUP); nGroup++)
```

```
        {
            for (int nJ = 0; nJ < SIZE(YOB_START15); nJ++)
            {
                if (nExpected[nJ][nGroup] > nRecorded[nJ][nGroup])
                {
                    for (int nI = 0; nI < nExpected[nJ][nGroup] - nRecorded[nJ][nGroup]; nI++)
                    {
                        if (asChildlessWomen[RANGE_POS(ALL_YEAR_RANGE, MIN(YOB_START15) +
nJ)][nGroup]->Count() > 0)
                        {
                            auto prMother = asChildlessWomen[RANGE_POS(ALL_YEAR_RANGE,
MIN(YOB_START15) + nJ)][nGroup]->GetRandom(RandUniform(19));
                            prMother->is_mother = TRUE;
                        }
                    }
                }
            }
        }
    }
}

table Person TabChildlessAtStartTab //EN Childlessness
[sex==FEMALE && calendar_year == 2010]
{
    {
        duration(is_mother,TRUE)/duration()
    }
    * integer_age
    * educ_fate +
};
```

Update of a version specific age-range in _CountryContext.mpp

```
range YOB_BIRTH1 { 1960, 2050 };                //EN Year of birth
range YOB_START15{ 1900, 1994 };               //EN Year of birth (in startpop age 15+)
range YOB_START50{ 1900, 1959 };               //EN Year of birth (in startpop age 50+)
//
```

Step 9: Male Childlessness

Model Description

At this step we add a module for setting the fate of male childlessness. Based on information from the starting population and a cohort parameter by education, lifetime childlessness is decided at the begin of the simulation or - for those born in the simulation - early in life before entering fertile age.

The new MaleChildlessness.mpp Module

This module decides the lifetime-fate of male childlessness based on two pieces of information: (1) living with children in the starting population file, and (2) a parameter of male cohort childlessness by education fate. Men currently living with children are assumed to be parents, thus not childless. In a second step, men from the starting population who are currently not living with children a status is assigned randomly in order to meet the cohort targets. For those born in the simulation, the status is assigned by sampling according the cohort parameter. This assignment happens at the first birthday, as at this time the final education fate is known.

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Sets
////////////////////////////////////////////////////////////////////////////////////////////////
//

//EN Men fated to stay childless by year of birth and education
actor_set Person asMenNeverFather[year_of_birth][birth1_group] filter never_father && sex == MALE;

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN Male cohort childlessness
    double MaleCohortChildlessness[YOB_MALEFERT][BIRTH1_GROUP];
};

parameter_group PG_MaleChildlessness        //EM Male Childlessness
{
    MaleCohortChildlessness
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states and functions
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    logical never_father = { TRUE };           //EN Person fated never to become father
    void    SetEverFatherFate();               //EN Set fate of ever becoming father
    hook    SetEverFatherFate, BirthdayEvent;  //EN Decide at 1st birthday (education fate known)
};

actor Clock
{
    void ImputeEverFather();                   //EN Person is father
    hook ImputeEverFather, ImputeIsMother;     //EN To be run immediately after female imputation
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void Person::SetEverFatherFate()
{
    if (person_type != PT_START && integer_age == 1 && sex==MALE)
    {
        never_father = (RandUniform(21)
            < MaleCohortChildlessness[RANGE_POS(YOB_MALEFERT, year_of_birth)][birth1_group]);
    }
}

void Clock::ImputeEverFather()
{
    if (clock_year == MIN(SIM_YEAR_RANGE))
    {
        int nPopSize;
        int nPersons[SIZE(YOB_START15)][SIZE(BIRTH1_GROUP)];
        int nExpected[SIZE(YOB_START15)][SIZE(BIRTH1_GROUP)];
        int nRecorded[SIZE(YOB_START15)][SIZE(BIRTH1_GROUP)];
        // initialize with 0
        for (int nJ = 0; nJ < SIZE(YOB_START15); nJ++)
        {
            for (int nGroup = 0; nGroup < SIZE(BIRTH1_GROUP); nGroup++)
            {
                nPersons[nJ][nGroup] = 0; nExpected[nJ][nGroup] = 0; nRecorded[nJ][nGroup] = 0;
            }
        }
        // assign status is_father where there are children in hh
        // count men and known fathers by yob
        nPopSize = asAllPersons->Count();
        for (int nI = 0; nI < nPopSize; nI++)
        {
            auto prPerson = asAllPersons->Item(nI);
            if (WITHIN(YOB_START15, prPerson->year_of_birth) && prPerson->sex == MALE)
            {
                int nYOB = prPerson->year_of_birth - MIN(YOB_START15);
                nPersons[nYOB][prPerson->birth1_group] = nPersons[nYOB][prPerson->birth1_group] +
1;
                if (prPerson->linked_kids > 0 || (prPerson->lSpouse && prPerson->lSpouse-
>linked_kids >0))
                {
                    nRecorded[nYOB][prPerson->birth1_group] = nRecorded[nYOB][prPerson-
>birth1_group] + 1;
                    prPerson->never_father = FALSE;
                }
            }
```

```
            else if (prPerson->sex == MALE) // those below 15
            {
                prPerson->never_father = (RandUniform(20) <
MaleCohortChildlessness[RANGE_POS(YOB_MALEFERT, prPerson->year_of_birth)][prPerson-
>birth1_group]);
            }
        }
        //calculate expected number of fathers by yob
        for (int nGroup = 0; nGroup < SIZE(BIRTH1_GROUP); nGroup++)
        {
            for (int nJ = 0; nJ < SIZE(YOB_START15); nJ++)
            {
                nExpected[nJ][nGroup] = round(nPersons[nJ][nGroup] * (1 -
MaleCohortChildlessness[RANGE_POS(YOB_MALEFERT, nJ+MIN(YOB_START15))][nGroup]));
            }
        }
        //randomly choose childless men to (ever) become fathers in order to meet target numbers
        for (int nGroup = 0; nGroup < SIZE(BIRTH1_GROUP); nGroup++)
        {
            for (int nJ = 0; nJ < SIZE(YOB_START15); nJ++)
            {
                if (nExpected[nJ][nGroup] > nRecorded[nJ][nGroup])
                {
                    for (int nI = 0; nI < nExpected[nJ][nGroup] - nRecorded[nJ][nGroup]; nI++)
                    {
                        if (asMenNeverFather[RANGE_POS(ALL_YEAR_RANGE, MIN(YOB_START15) +
nJ)][nGroup]->Count() > 0)
                        {
                            auto prFather = asMenNeverFather[RANGE_POS(ALL_YEAR_RANGE,
MIN(YOB_START15) + nJ)][nGroup]->GetRandom(RandUniform(19));
                            prFather->never_father = FALSE;
                        }
                    }
                }
            }
        }
    }
 }

table Person TabIsFather //EN Is ever Father
[sex == MALE && calendar_year > 2010 && integer_age>1]
{
    {
        duration(never_father,FALSE) / duration()
    }
    * year_of_birth
    * educ_fate +
};
```

Update of a version specific age-range in _CountryContext.mpp

```
range YOB_MALEFERT{ 1920, 2050 };              //EN Year of birth
```

Step 10: Education Pattern

Model Description

At this step we add a module for education pattern: current school attendance, school type, and fulltime/part-time status. Pattern are sampled for a given highest educational attainment 'fate' and gender.

The new EducationPattern.mpp Module

The module implements school patterns of school attendance, school type, and enrolment type for a given education fate. School types correspond to the levels low, medium, high but additionally differentiate between dual, vocational, and general tracks. Attendence can be fulltime or parttime. The model is a 'fate-model' where for a given final outcome a pattern is assigned early in life. Patterns depend on final outcome (which itself depends on year of birth, sex and parents's characteristics) and sex. For people of the starting population, educational pattern are imputed using the same model, but if possible respecting the variables on current school attendance. This is archieved in two steps. First for a given fate a pattern is assigned. When all perspms of a birth cohort have their fates assigned, these fates and pattern can be 'traded' with others in order to match the school attendance status in the starting population. These trades are within groups by sex and parents' education thus don not affect the aggregate **modeled** school attendance.

Once assigned, the actual school attendance pattern are then updated in yearly schoolyear steps. For a given final fate and education pattern, three states are set:

- educ_level: the current highest education attainment
- educ_status: the current education status (ful-time, part-time, dual, pause etc.)
- educ_pattern_status: the current school type and attendence pattern (place in assigned pattern)

```
//TODO check if for all who need it parents education is available

////////////////////////////////////////////////////////////////////////////////
/
// Dimensions
////////////////////////////////////////////////////////////////////////////////
/

//TODO set exact years, move to _CountryContext
//EN Year of birth for which education patterns have to be checked against school attendance
status in startpop
```

```
range YOB_CHECK_SCHOOL{ 1980,1990 };

classification EDUC_STATUS                              //EN Education Status
{
    ES_NEVER,                                           //EN Not entered school
    ES_FULLTIME,                                        //EN Fulltime Student
    ES_PARTTIME,                                        //EN Parttime Student
    ES_DUAL,                                            //EN Dual system student
    ES_PAUSE,                                           //EN Pause (interruption e.g. military)
    ES_FIN                                              //EN Finished schooling
};

classification EDUC_PATTERN                             //EN Education pattern
{
    EP_LOW,                                             //EN Low
    EP_MED_DUAL,                                        //EN Medium Dual
    EP_MED_VOC,                                         //EN Medium Vocational
    EP_MED_GEN,                                         //EN Medium General
    EP_OUT1,                                            //EN Out of school spell 1
    EP_HIGH1_FT,                                        //EN High Episode 1 (Full-time)
    EP_HIGH1_PT,                                        //EN High Episode 1 (Part-time)
    EP_OUT2,                                            //EN Out of school spell 2
    EP_HIGH2_FT,                                        //EN High Episode 2 (Full-time)
    EP_HIGH2_PT,                                        //EN High Episode 2 (Part-time)
    EP_OUT3,                                            //EN Out of school spell 3
    EP_HIGH3_FT,                                        //EN High Episode 3 (Full-time)
    EP_HIGH3_PT                                         //EN High Episode 3 (Part-time)
};

range EDUC_PATTERN_RANGE{ 0, 11 };                      //EN Education Pattern

////////////////////////////////////////////////////////////////////////////////////////////////
/
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////////
/

parameters
{
    //EN Education Pattern
    int        EducPattern[EDUC_LEVEL3][EDUC_PATTERN_RANGE][EDUC_PATTERN];

    //EN Education Pattern Distribution
    cumrate[1] EducPatternDist[SEX][EDUC_LEVEL3][EDUC_PATTERN_RANGE];

    int        SchoolEntryAge;                          //EN School entry age
    double     StartSchoolYear;                         //EN Start of school year
};

parameter_group PG_Education                            //EN Education
{
    SchoolEntryAge, StartSchoolYear,
    EducPattern, EducPatternDist
};

////////////////////////////////////////////////////////////////////////////////////////////////
/
// Actor sets
////////////////////////////////////////////////////////////////////////////////////////////////
/

//EN Actor set of all potential students
actor_set Person asAllPotentialStudents
    filter is_alive && integer_age >= SchoolEntryAge && year_finish_school >= calendar_year;

//EN Persons who study in startpop and have no matching fate
```

```
actor_set Person asWantTradeEducPatternToInSchool[educ_group][sex]
    filter integer_age == 2 && want_trade_educ_pattern_to_inschool;

//EN Persons who do not study in startpop and have no matching fate
actor_set Person asWantTradeEducPatternToOutSchool[educ_group][sex]
    filter integer_age == 2 && want_trade_educ_pattern_to_outschool;

////////////////////////////////////////////////////////////////////////////////////////////////
/
// Actor declarations
////////////////////////////////////////////////////////////////////////////////////////////////
/

actor Person
{
    EDUC_LEVEL5 educ_level = { EL5_LOW };               //EN Current education level
    EDUC_STATUS educ_status = { ES_NEVER };             //EN Current Education Status
    logical in_school_startpop = { FALSE };             //EN Attending school in starting
population

    int     year_finish_school  = { 9999 };             //EN Year finishing school

    EDUC_PATTERN_RANGE educ_pattern_number = { 0 };     //EN Educ Pattern Number
    EDUC_PATTERN educ_pattern_status = { EP_LOW };      //EN Education pattern status

    void SampleEducPattern();                           //EN Sample education fate and pattern
    hook SampleEducPattern, YearEnd;

    //EN Set Education states for given spell
    void SetCurrentEducLevelPatternStatus(int nSchoolSpell);

    void SchoolYearChange();                            //EN School year change

    logical want_trade_educ_pattern_to_inschool  = { FALSE };
    logical want_trade_educ_pattern_to_outschool = { FALSE };
};

actor Clock
{
    void SetNextSchoolYear();
    hook SetNextSchoolYear, StartYearClock;

    void TradeEducationFatesAndPatterns();
    hook TradeEducationFatesAndPatterns, StartYearClock;

    TIME    next_school_year = { TIME_INFINITE };       //EN Next School Year
    event   timeSchoolYearClock, SchoolYearClock;       //EN School year clock event
};

////////////////////////////////////////////////////////////////////////////////////////////////
/
// Implementation Clock Functions
////////////////////////////////////////////////////////////////////////////////////////////////
/


TIME Clock::timeSchoolYearClock() { return next_school_year; }
void Clock::SchoolYearClock()
{
    long nPopSize = asAllPotentialStudents->Count();
    for (long nIndex = 0; nIndex < nPopSize; nIndex++)
    {
        Person *prPerson = asAllPotentialStudents->Item(nIndex);
        prPerson->SchoolYearChange();
    }
    next_school_year = TIME_INFINITE;
```

```
}

void Clock::TradeEducationFatesAndPatterns()
{
    for (int nGroup = 0; nGroup < SIZE(EDUC_GROUP); nGroup++)
    {
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            while (asWantTradeEducPatternToInSchool[nGroup][nSex]->Count() > 0 &&
asWantTradeEducPatternToOutSchool[nGroup][nSex]->Count() > 0)
            {
                auto prPersonA = asWantTradeEducPatternToInSchool[nGroup][nSex]-
>GetRandom(RandUniform(24));
                auto prPersonB = asWantTradeEducPatternToOutSchool[nGroup][nSex]-
>GetRandom(RandUniform(25));

                EDUC_LEVEL3 cFate = prPersonA->educ_fate;
                EDUC_PATTERN_RANGE cPattern = prPersonA->educ_pattern_number;

                prPersonA->educ_fate = prPersonB->educ_fate;
                prPersonA->educ_pattern_number = prPersonB->educ_pattern_number;
                prPersonA->want_trade_educ_pattern_to_inschool = FALSE;

                prPersonB->educ_fate = cFate;
                prPersonB->educ_pattern_number = cPattern;
                prPersonB->want_trade_educ_pattern_to_outschool = FALSE;
            }
        }
    }
}

void Clock::SetNextSchoolYear()
{
    next_school_year = WAIT(StartSchoolYear); // set school year clock
}

////////////////////////////////////////////////////////////////////////////////////////////////
/
// Implementation Person Functions
////////////////////////////////////////////////////////////////////////////////////////////////
/

void Person::SampleEducPattern()
{
    if (integer_age == 1)
    {
        int nPattern;
        Lookup_EducPatternDist(RandUniform(6), (int)sex, (int)educ_fate, &nPattern);
        educ_pattern_number = (EDUC_PATTERN_RANGE)nPattern;
        // checks if in_school_startpop contradicts the fate
        if (person_type == PT_START && WITHIN(YOB_CHECK_SCHOOL, year_of_birth))
        {
            int     nSchoolTerm = MIN(SIM_YEAR_RANGE) - year_of_birth - SchoolEntryAge - 1;
            SetCurrentEducLevelPatternStatus(nSchoolTerm);
            if (educ_status == ES_FULLTIME || educ_status == ES_PARTTIME || educ_status ==
ES_DUAL)
            {
                if (!in_school_startpop) want_trade_educ_pattern_to_outschool = TRUE;
            }
            else if (in_school_startpop) want_trade_educ_pattern_to_inschool = TRUE;
        }
    }
}

void Person::SetCurrentEducLevelPatternStatus(int nSchoolSpell)
{
```

```
EDUC_PATTERN    cPreviousPatternStatus = educ_pattern_status; //EN Store the current status

//Calculate maximal number of spells for a given educ_fate and educ_pattern_number
int nSumSpells = 0;
for (int nIndex = 0; nIndex < SIZE(EDUC_PATTERN); nIndex++)
{
    nSumSpells = nSumSpells + EducPattern[educ_fate][educ_pattern_number][nIndex];
}

if (nSchoolSpell == 0)                  // Not Started School
{
    educ_status = ES_NEVER;
    educ_level  = EL5_LOW;
}
else if (nSchoolSpell > nSumSpells)     // Beyond max spells
{
    educ_status = ES_FIN;               // Finished school
}
else                                    // Somewhere in system
{
    // Find current educ_pattern_status
    bool bFound = FALSE;  int nColumn = 0;  int nItem = 0;  int nCount = 0;
    while (!bFound)
    {
        nItem++; nCount++;
        while (nItem > EducPattern[educ_fate][educ_pattern_number][nColumn])
        {
            nItem = 1; nColumn++;
        }
        if (nCount == nSchoolSpell)     // found current educ_pattern_status
        {
            bFound = TRUE;
            educ_pattern_status = ( EDUC_PATTERN )nColumn;
        }
    }
    // Update educ_status
    if (educ_pattern_status == EP_HIGH1_PT || educ_pattern_status == EP_HIGH2_PT
        || educ_pattern_status == EP_HIGH3_PT )
    {
        educ_status = ES_PARTTIME;
    }
    else if (educ_pattern_status == EP_MED_DUAL )
    {
        educ_status = ES_DUAL;
    }
    else if (educ_pattern_status == EP_OUT1 || educ_pattern_status == EP_OUT2
        || educ_pattern_status == EP_OUT3)
    {
        educ_status = ES_PAUSE;
    }
    else educ_status = ES_FULLTIME;
}

// Update current education level
if (educ_status == ES_FIN) //End of studies reached
{
    EDUC_LEVEL5 elEducLevel;
    if (educ_fate == EL3_LOW)
    {
        elEducLevel = EL5_LOW;
    }
    else if (educ_fate == EL3_HIGH)
    {
        elEducLevel = EL5_HIGH;
    }
```

```
        else if (educ_fate == EL3_MEDIUM && educ_level == EL5_LOW && cPreviousPatternStatus ==
EP_MED_DUAL)
        {
            elEducLevel = EL5_MEDIUMD;
        }
        else if (educ_fate == EL3_MEDIUM && educ_level == EL5_LOW && cPreviousPatternStatus ==
EP_MED_VOC)
        {
            elEducLevel = EL5_MEDIUMV;
        }
        else if (educ_fate == EL3_MEDIUM && educ_level == EL5_LOW && cPreviousPatternStatus ==
EP_MED_GEN)
        {
            elEducLevel = EL5_MEDIUMG;
        }
        else elEducLevel = educ_level;
        educ_level = elEducLevel;
    }
    // Change of school type to high or pause (must have finished medium)
    else if (educ_level == EL5_LOW && (educ_status == ES_PAUSE
        || educ_pattern_status == EP_HIGH1_FT || educ_pattern_status == EP_HIGH1_PT))
    {
        if (cPreviousPatternStatus == EP_MED_DUAL) educ_level = EL5_MEDIUMD;
        else if (cPreviousPatternStatus == EP_MED_VOC) educ_level = EL5_MEDIUMV;
        else if (cPreviousPatternStatus == EP_MED_GEN) educ_level = EL5_MEDIUMG;
    }
}

void Person::SchoolYearChange()
{
    if (integer_age >= SchoolEntryAge) SetCurrentEducLevelPatternStatus(integer_age -
SchoolEntryAge + 1);
    if (educ_status == ES_FIN && year_finish_school > calendar_year) year_finish_school =
calendar_year;
}

table Person TabEducTrade2  //EN TEST Trading school age 2
[integer_age == 2]
{
    educ_group + *
    sex + *
{
    duration(want_trade_educ_pattern_to_inschool,TRUE) / duration(), //EN Wants trade to inschool
decimals=4
        duration(want_trade_educ_pattern_to_outschool,TRUE) / duration() //EN Wants trade to
outschool decimals=4
}
*year_of_birth
};

table Person TabEducTrade3  //EN TEST Trading school age 3
[integer_age == 4]
{
    educ_group + *
    sex + *
    {
        duration(want_trade_educ_pattern_to_inschool,TRUE) / duration(), //EN Wants trade to
inschool decimals=4
        duration(want_trade_educ_pattern_to_outschool,TRUE) / duration() //EN Wants trade to
outschool decimals=4
    }
    *year_of_birth
};

table Person TabEducParents2  //EN TEST Education of parents
[integer_age == 2]
```

```
{
{
        duration(educ_parents,EL3_LOW) / duration(), //EN LOW decimals=4
        duration(educ_parents,EL3_MEDIUM) / duration(), //EN MED decimals=4
        duration(educ_parents,EL3_HIGH) / duration() //EN HIG decimals=4
}
*year_of_birth
};
```

Initialisations in the Start() function in PersonCore.mpp

The state in_school_startpop os initialized in the Start function for those coming from the starting population.

```
    // Initialize states
    if (person_type == PT_START) // Person comes from starting population
    {
        // (A) States from Starting population file
        time                = peObservation->pmc[PMC_BIRTH] + RandUniform(2);
        sex                 = (SEX)(int)peObservation->pmc[PMC_SEX];
        family_role         = (FAM_ROLE)( int )peObservation->pmc[PMC_ROLE];
        educ_startpop       = (EDUC_LEVEL5)(int)peObservation->pmc[PMC_EDUC];
        in_school_startpop  = (bool)(int)peObservation->pmc[PMC_INSCHOOL];
//
```

Step 11: Female Partnership Status

Overview

At this step we add a new module for maintaining the female partnership status according to observed partnership patterns by age (at last birth), age of youngest child, and education. The female partnership status is updated at the middle of each year according to model parameters. This is a base module working entirely due to alignment. It can be refined by adding union dissolution events on the micro level.

The new FemalePartnershipStatus.mpp Module

This module implements processes for maintaining the partnership status of women over the life course (union formation, disslolution, calls for matching a suitable partner). The female partnership status is updated at yearly events according to observed partnership patterns by education, age / age group at last birth, and age group of youngest child. The partnership status is modeled for all women within the age range 15-80, no more union formation events are modled thereafter when it is assumed the onlz union dissolution is due to widowhood.

Parameters:

- Proportion of women living with dependent children who are in a partnership by education, age group at last birth and age group of youngest child
- Proportion of women not living with dependent children who are in a partnership by education and age

The model maintaines the patterns contained in the parameters in the future, thus we assume that these patterns are stable and changes in aggregate partnership characteristics only result from compositional changes in the female population like childlessness and timing of births. The model follows a ‘minimum necessary corrections’ approach changing the union status of women only to meet aggregate numbers. In reality, unions are more unstable, i.e. the model does not move women out of a union and others in if the aggregate proportion does not change. It can be refined e.g. by adding a union dissolution module at the micro level if a higher life course consistency is important for model applications.

The core of this module is the Clock function UpdatePartnershipStatus() which is hooked to the Clock’s mid-year event. Alternatively, the function could be hooked to the mid-month event to add precision (at the cost of performance).

Code Changes in other modules:

- The module requires a function FindPartner() which is declared and implemented in the module PartnerMatching.mpp.

```
link Person.lMother   Person.mlMothersChildren[];                    //EN Link between children
and Mother
link Person.lFather   Person.mlFathersChildren[];                    //EN Link between children
and Father

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Sets
////////////////////////////////////////////////////////////////////////////////////////////
//

//EN Women in a partnership living with dependent children
actor_set Person asPopInUnionWithChildren[educ3_level][child_agegr][moth_agegr]
    filter is_alive && sex == FEMALE && in_projected_time && WITHIN(PART_AGE_RANGE, integer_age)
&&
    in_union && lives_with_dependent_child;

//EN Women not in a partnership living with dependent children
actor_set Person asPopNotInUnionWithChildren[educ3_level][child_agegr][moth_agegr]
    filter is_alive && sex == FEMALE && in_projected_time && WITHIN(PART_AGE_RANGE, integer_age)
&&
    !in_union && lives_with_dependent_child;

//EN Women living with dependent children
actor_set Person asPopWithChildren[educ3_level][child_agegr][moth_agegr]
    filter is_alive && sex == FEMALE && in_projected_time && WITHIN(PART_AGE_RANGE, integer_age)
&&
    lives_with_dependent_child;

//EN Women not living with dependent children
actor_set Person PopNoChildren[educ3_level][integer_age]
    filter is_alive && sex == FEMALE && in_projected_time && WITHIN(PART_AGE_RANGE, integer_age)
&&
    !lives_with_dependent_child;

//EN Women in a partnership not living with dependent children
actor_set Person asPopInUnionNoChildren[educ3_level][integer_age]
    filter is_alive && sex == FEMALE && in_projected_time && WITHIN(PART_AGE_RANGE, integer_age)
&&
    in_union && !lives_with_dependent_child;

//EN Women not in a partnership not living with dependent children
actor_set Person asPopNotInUnionNoChildren[educ3_level][integer_age]
    filter is_alive && sex == FEMALE && in_projected_time && WITHIN(PART_AGE_RANGE, integer_age)
&&
    !in_union && !lives_with_dependent_child;


////////////////////////////////////////////////////////////////////////////////////////////
//
// Dimensions
////////////////////////////////////////////////////////////////////////////////////////////
//

partition   CHILD_AGEGR_PART { 1,3,6,9,12,15 };        //EN Age of youngest child
partition   MOTH_AGEGR_PART { 20, 25, 30, 35, 40 };    //EN Age of mother at last birth
```

```
range       PART_AGE_RANGE { 15, 80 };            //EN Age

classification MOTH_AGEGR //EN Age group mothers at birth
{
    CMA20,  //EN Below 20
    CMA25,  //EN 20 to 24
    CMA30,  //EN 25 to 19
    CMA35,  //EN 30 to 34
    CMA40,  //EN 35 to 39
    CMA40P  //EN 40+
};

classification CHILD_AGEGR //EN Age group child
{
    CA00,  //EN 0
    CA01,  //EN 1 to 2
    CA03,  //EN 3 to 5
    CA06,  //EN 6 to 8
    CA09,  //EN 9 to 11
    CA12,  //EN 12 to 14
    CA15   //EN 15 to 17
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN Probability to be in a partnership - Females living with children
    double      InUnionProbWithChildren[EDUC_LEVEL3][CHILD_AGEGR][MOTH_AGEGR];

    //EN Probability to be in a partnership - Females not living with children
    double      InUnionProbNoChildren[PART_AGE_RANGE][EDUC_LEVEL3];
};

parameter_group PG_FemalePartnerships           //EN Female Partnership Status
{
    InUnionProbWithChildren,
    InUnionProbNoChildren
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor States & Functions
////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    int int_age = integer_age;
    EDUC_LEVEL3 educ3_level = aggregate(educ_level, EDUC_LEVEL3);

    //EN Person currently in a union
    logical in_union = (lSpouse != NULL) ? TRUE : FALSE;

    //EN Age of youngest child of women
    double  age_youngest_child = (count(mlMothersChildren) > 0 && sex == FEMALE) ?
        min_over(mlMothersChildren, int_age) : TIME_INFINITE;

    //EN Age group of youngest child of women
    int child_agegr_part = split(age_youngest_child, CHILD_AGEGR_PART);

    //EN Woman lives with a child afe < 18
```

```
    logical lives_with_dependent_child = (age_youngest_child < 18) ? TRUE : FALSE;

    //EN Woman's age at last birth if living with children < 18
    double age_last_birth = (lives_with_dependent_child) ?
        integer_age - age_youngest_child : TIME_INFINITE;

    //EN Age group at last birth
    int moth_agegr_part = split(age_last_birth, MOTH_AGEGR_PART);

    //EN Age group at last birth
    MOTH_AGEGR moth_agegr = (moth_agegr_part == 0) ? CMA20 :
        (moth_agegr_part == 1) ? CMA25 :
        (moth_agegr_part == 2) ? CMA30 :
        (moth_agegr_part == 3) ? CMA35 :
        (moth_agegr_part == 4) ? CMA40 : CMA40P;

    //EN Age group child
    CHILD_AGEGR child_agegr = (child_agegr_part == 0) ? CA00 :
        (child_agegr_part == 1) ? CA01 :
        (child_agegr_part == 2) ? CA03 :
        (child_agegr_part == 3) ? CA06 :
        (child_agegr_part == 4) ? CA09 :
        (child_agegr_part == 5) ? CA12 : CA15;
};

actor Clock
{
    void UpdatePartnershipStatus();                     //EN Update Female Partnership Status
    hook UpdatePartnershipStatus, MidYearClockEvent;    //EN hook to mid year
};

void Clock::UpdatePartnershipStatus()
{
    long nTarget;
    if (clock_year >= MIN(SIM_YEAR_RANGE))
    {
        // Women with children
        for (int nEduc = 0; nEduc < SIZE(EDUC_LEVEL3); nEduc++)
        {
            for (int nChildAge = 0; nChildAge < SIZE(CHILD_AGEGR); nChildAge++)
            {
                for (int nMothAge = 0; nMothAge < SIZE(MOTH_AGEGR); nMothAge++)
                {
                    nTarget = round(InUnionProbWithChildren[nEduc][nChildAge][nMothAge] *
                        asPopWithChildren[nEduc][nChildAge][nMothAge]->Count());

                    if (nTarget > asPopInUnionWithChildren[nEduc][nChildAge][nMothAge]->Count())
                    {
                        long nEmptyRun = 0;
                        while (nTarget > asPopInUnionWithChildren[nEduc][nChildAge][nMothAge]-
>Count() &&
                            asPopNotInUnionWithChildren[nEduc][nChildAge][nMothAge]->Count() > 0
&& nEmptyRun < 10000)
                        {
                            auto prFam = asPopNotInUnionWithChildren[nEduc][nChildAge][nMothAge]-
>GetRandom(RandUniform(55));
                            if (!prFam->FindPartner()) nEmptyRun++;
                        }
                    }
                    else if (nTarget < asPopInUnionWithChildren[nEduc][nChildAge][nMothAge]-
>Count())
                    {
                        while (nTarget < asPopInUnionWithChildren[nEduc][nChildAge][nMothAge]-
>Count() &&
                            asPopInUnionWithChildren[nEduc][nChildAge][nMothAge]->Count() > 0)
                        {
```

```
                            auto prFam = asPopInUnionWithChildren[nEduc][nChildAge][nMothAge]-
>GetRandom(RandUniform(57));
                            prFam->lSpouse = NULL;
                        }
                    }
                }
            }
        }
        //Targets for women without children in hh
        for (int nEduc = 0; nEduc < SIZE(EDUC_LEVEL3); nEduc++)
        {
            for (int nAge = 0; nAge < SIZE(PART_AGE_RANGE); nAge++)
            {
                nTarget = round(InUnionProbNoChildren[nAge][nEduc] *
                    PopNoChildren[nEduc][nAge + MIN(PART_AGE_RANGE)]->Count());

                if (nTarget > asPopInUnionNoChildren[nEduc][nAge + MIN(PART_AGE_RANGE)]->Count())
                {
                    long nEmptyRun = 0;
                    while (nTarget > asPopInUnionNoChildren[nEduc][nAge + MIN(PART_AGE_RANGE)]-
>Count() &&
                        asPopNotInUnionNoChildren[nEduc][nAge + MIN(PART_AGE_RANGE)]->Count() > 0
&& nEmptyRun < 10000)
                    {
                        auto prFam = asPopNotInUnionNoChildren[nEduc][nAge + MIN(PART_AGE_RANGE)]-
>GetRandom(RandUniform(58));
                        if (!prFam->FindPartner()) nEmptyRun++;
                    }
                }
                else if (nTarget < asPopInUnionNoChildren[nEduc][nAge + MIN(PART_AGE_RANGE)]-
>Count())
                {
                    while (nTarget < asPopInUnionNoChildren[nEduc][nAge + MIN(PART_AGE_RANGE)]-
>Count() &&
                        asPopInUnionNoChildren[nEduc][nAge + MIN(PART_AGE_RANGE)]->Count() > 0)
                    {
                        auto prFam = asPopInUnionNoChildren[nEduc][nAge + MIN(PART_AGE_RANGE)]-
>GetRandom(RandUniform(61));
                        prFam->lSpouse = NULL;
                    }
                }
            }
        }
    }
}


////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TabValidationPartnershipStatus  //EN Female partnership status
{
    TabTestPartnershipStatusWithKids2020,
    TabTestPartnershipStatusNoKids2020,
    TabParaPartnershipStatusWithKids,
    TabParaPartnershipStatusNoKids
};

table Person TabTestPartnershipStatusWithKids2020
[lives_with_dependent_child && in_projected_time && calendar_year == 2020
&& sex == FEMALE && WITHIN(PART_AGE_RANGE, integer_age)]
{
    educ3_level + *
```

```
{
    duration(in_union,TRUE) / duration() //EN Proportion in partnership decimals=4
}
*moth_agegr +
*child_agegr +
};

table Person TabTestPartnershipStatusNoKids2020
[!lives_with_dependent_child && in_projected_time && calendar_year == 2020
&& sex == FEMALE && WITHIN(PART_AGE_RANGE, integer_age)]
{
    {
        duration(in_union, TRUE) / duration() //EN Proportion in partnership decimals=4
    }
    *integer_age +
    *educ3_level +
};


////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables for parameter generation

partition SECOND_AFTER{ 2010.01, 2010.02 };
actor Person
{
    logical second_after = (self_scheduling_split(time,SECOND_AFTER) == 1) ? TRUE : FALSE;
};

//EN Proportion of women living with dependent children who are in a partnership
table Person TabParaPartnershipStatusWithKids
[trigger_entrances(second_after, TRUE) && lives_with_dependent_child && in_projected_time
&& sex == FEMALE && WITHIN(PART_AGE_RANGE, integer_age)]
{
    educ3_level + *
{
    value_in(in_union) / unit //EN Proportion in partnership decimals=4
}
*child_agegr +
*moth_agegr +
};

//EN Proportion of women not living with dependent children who are in a partnership
table Person TabParaPartnershipStatusNoKids
[trigger_entrances(second_after, TRUE) && !lives_with_dependent_child && in_projected_time
&& sex == FEMALE && WITHIN(PART_AGE_RANGE, integer_age)]
{
    {
        value_in(in_union) / unit //EN Proportion in partnership decimals=4
    }
    *integer_age +
    *educ3_level +
};
```

The new module PartnerMatching.mpp

At the current step 11 of model development, this module is a dummy module only. It implements the function FindPartner which randomly chooses and links an available partner

of the same age. If non is found, the function returns FALSE. This module is to be refined at a later step.

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Sets
////////////////////////////////////////////////////////////////////////////////////////////////
//

//EN Males not in a partnershio by age
actor_set Person asAvailableMale[integer_age] filter is_alive && sex == MALE && !lSpouse;

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states, events and functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    logical FindPartner();
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

logical Person::FindPartner()
{
    if (asAvailableMale[integer_age]->Count() > 0)
    {
        auto prPartner = asAvailableMale[integer_age]->GetRandom(RandUniform(26));
        lSpouse = prPartner;
        return TRUE;
    }
    else return FALSE;
}
```

Step 12: Partner Matching

Overview

At this step we refine the module PartnerMatching.mpp. Partners are matched by distributional tables by age and education. Also, the model accounts for the consequences of childlessness on partnership bahavior.

The refined PartnerMatching.mpp Module

Partner matching is modeled by age, education and childlessness. Currently we only model heterosexual couples. Concerning age differences, we follow a “photo approach”, meaning that we assume that the patterns in observed age differences of couples by age persist over time. One difficulty in assigning a partner lies in the changing distribution of age differences by age of union formation. For example, a young man cannot have a much younger spouse (or vice versa), while the spread of observed age differences is widening with age. As the information on union duration is typically not available in most surveys, and admin data exist only for marriages, we follow an indirect approach oriented on the observed age patterns in existing partnerships. The algorithm is as follows:

- Based on a parameter on the distribution of the age differences of couples for a given age of the searching female and the cohort size of women in partnerships in the simulation, we calculate the expected number of partners by age.
- We then calculate the number of existing partners by age in the simulated population.
- By comparing the expected with the observed numbers, we identify the age with the largest negative gap for which at least one available male spouse is available.

After having identified the pool of available partners, a second criterion is education. In contrast to age, we use the education patterns of currently young couples (age 25-45) for deciding on the spouse’s education. It is assumed, that current patterns are quite persistent and maintainable over time. If no match is possible for a draw based on these probabilities, the experiment is repeated; in contrast, if more than one man stays in the pool, the search moves to its third step.

The remaining criterium is male cohort childlessness. If the pool contains both men flagged as cohort childless and men who are supposed to become fathers at some point in time, it is the latter who are chosen first. To maintain consistency concerning male childlessness, in the case of birth the cohort childless flag is tried to be passed to an un-flagged single male of the same age and sex.

This module is highly experimental. At each step, it can be tracked if a match can be made or the experiment must be repeated. Refinements are possible especially concerning the order of the applied selection criteria which can be changed or follow a random distribution.

```
link Person.lPartner;                                   //EN Link between spouses

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Sets
////////////////////////////////////////////////////////////////////////////////////////////
//

//EN Males not in a partnership by age
actor_set Person asAvailableMale[integer_age]
    filter is_alive && sex == MALE && !has_partner && in_projected_time
    && WITHIN(MALE_PART_AGE_RANGE, integer_age);

//EN Males not in a partnership by age and education
actor_set Person asAvailableMaleEduc[integer_age][educ_fate]
    filter is_alive && sex == MALE && !has_partner && in_projected_time
    && WITHIN(MALE_PART_AGE_RANGE, integer_age);

//EN Males not in a partnership by age, education and cohort childlessness
actor_set Person asAvailableMaleEducChildless[integer_age][educ_fate][never_father]
    filter is_alive && sex == MALE && !has_partner && in_projected_time
    && WITHIN(MALE_PART_AGE_RANGE, integer_age);

//EN Women in a partnership
actor_set Person asFamInUnion[integer_age][integer_age_partner]
    filter is_alive && sex == FEMALE && in_projected_time && WITHIN(PART_AGE_RANGE, integer_age)
    && has_partner;

//EN Single males not never_father
actor_set Person asSingleMaleEverFatherEduc[integer_age][educ_fate]
    filter is_alive && sex == MALE && !has_partner && !never_father && in_projected_time;

//EN Couple males not never_father
actor_set Person asCoupleMaleEverFatherEduc[integer_age][educ_fate]
    filter is_alive && sex == MALE && has_partner && !never_father && in_projected_time
&& !is_blocked;



////////////////////////////////////////////////////////////////////////////////////////////
//
// Dimensions
////////////////////////////////////////////////////////////////////////////////////////////
//

range       MALE_PART_AGE_RANGE { 15, 105 };                //EN Age

//MARS TEST
classification FATHER_TYPE                                  //EN Father type
{
    FT_NONO,        //EN No Father, never father
    FT_NOYES,       //EN No Father, could be father
    FT_YESNO,       //EN Father, never father
    FT_YESYES       //EN Father, can be father
};
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN Distribution of partner ages by age of female partner
    double PartnerAgeDistribution[PART_AGE_RANGE][MALE_PART_AGE_RANGE];

    //EN Partner Education
    cumrate PartnerEducation[EDUC_LEVEL3][EDUC_LEVEL3];
};


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states, events and functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    AGE_RANGE   integer_age_partner = (has_partner) ? lPartner->integer_age : 0;
    EDUC_LEVEL3 educ_partner = (has_partner) ? lPartner->educ_fate : EL3_LOW;

    logical     FindPartner();

    //EN Person currently in a union
    logical has_partner = (lPartner) ? TRUE : FALSE;

    void CheckFatherChildlessStatus(); hook CheckFatherChildlessStatus, MakeBaby;

    logical is_blocked = { FALSE };
    logical ever_father_in_sim = { FALSE }; //MARS TEST
    //MARS TEST FATHER TYPE
    FATHER_TYPE father_type =
        (!ever_father_in_sim && never_father) ? FT_NONO :
        (!ever_father_in_sim && !never_father) ? FT_NOYES :
        (ever_father_in_sim && never_father) ? FT_YESNO : FT_YESYES;
};


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//


void Person::CheckFatherChildlessStatus()
{
    if (has_partner && lPartner->never_father)
    {
        if (asSingleMaleEverFatherEduc[lPartner->integer_age][lPartner->educ_fate]->Count() > 0)
        {
            auto prPerson = asSingleMaleEverFatherEduc[lPartner->integer_age][lPartner-
>educ_fate]->GetRandom(RandUniform(30));
            prPerson->never_father = TRUE;
            lPartner->never_father = FALSE;
        }
        else if (asCoupleMaleEverFatherEduc[lPartner->integer_age][lPartner->educ_fate]->Count() >
1)
        {
```

```
            lPartner->is_blocked == TRUE;
            auto prPerson = asCoupleMaleEverFatherEduc[lPartner->integer_age][lPartner-
>educ_fate]->GetRandom(RandUniform(31));
            prPerson->never_father = TRUE;
            lPartner->never_father = FALSE;
            lPartner->is_blocked = FALSE;
        }
    }
    if (has_partner) lPartner->ever_father_in_sim = TRUE; //MARS TEST
}

logical Person::FindPartner()
{
    bool   bFoundSpouse = FALSE;
    double dExpectedPartners[SIZE(MALE_PART_AGE_RANGE)];
    double dObservedPartners[SIZE(MALE_PART_AGE_RANGE)];
    double dSumExpectedPartners = 0;
    double dSumObservedPartners = 0;
    double dGap = 0.0;
    double dLargestGap = 0.0;
    int    nAgePartner;
    int    nEducPartner;

    //(1) Partner Age
    for (long nI = 0; nI < SIZE(MALE_PART_AGE_RANGE); nI++)
    {
        dExpectedPartners[nI] = PartnerAgeDistribution[RANGE_POS(PART_AGE_RANGE,
integer_age)][nI];
        dObservedPartners[nI] = asFamInUnion[integer_age][MIN(MALE_PART_AGE_RANGE) + nI]->Count();

        dSumExpectedPartners = dSumExpectedPartners + dExpectedPartners[nI];
        dSumObservedPartners = dSumObservedPartners + dObservedPartners[nI];
    }
    for (long nI = 0; nI < SIZE(MALE_PART_AGE_RANGE); nI++)
    {
        if (dSumObservedPartners == 0.0) dSumObservedPartners = 1.0;
        dExpectedPartners[nI] = 1.001 * dExpectedPartners[nI] / dSumExpectedPartners;
        dObservedPartners[nI] = dObservedPartners[nI] / dSumObservedPartners;
        dGap = dExpectedPartners[nI] - dObservedPartners[nI];
        if (dExpectedPartners[nI] > 0.0 && dGap > dLargestGap &&
asAvailableMale[MIN(MALE_PART_AGE_RANGE)+nI] -> Count() > 0 )
        {
            dLargestGap = dGap;
            bFoundSpouse = TRUE;
            nAgePartner = MIN(MALE_PART_AGE_RANGE) + nI;
        }
    }
    //(2) Partner Education
    if (bFoundSpouse)
    {
        bool    bFoundSpouseByEduc = FALSE;
        int     nTrials = 15;
        while (!bFoundSpouseByEduc && nTrials > 0)
        {
            Lookup_PartnerEducation(RandUniform(27), (int)educ_fate, &nEducPartner);
            if (asAvailableMaleEduc[nAgePartner][nEducPartner]->Count() > 0)
            {
                bFoundSpouseByEduc = TRUE;
            }
            nTrials--;
        }
        if (!bFoundSpouseByEduc)
        {
            nEducPartner = asAvailableMale[nAgePartner]->GetRandom(RandUniform(28))->educ_fate;
        }
```

```
        //(3) Partner Childlessness
        if (asAvailableMaleEducChildless[nAgePartner][nEducPartner][FALSE]->Count() > 0)
        {
            lPartner = asAvailableMaleEducChildless[nAgePartner][nEducPartner][FALSE]-
>GetRandom(RandUniform(26));
        }
        else if (asAvailableMaleEducChildless[nAgePartner][nEducPartner][TRUE]->Count() > 0)
        {
            lPartner = asAvailableMaleEducChildless[nAgePartner][nEducPartner][TRUE]-
>GetRandom(RandUniform(29));
        }
    }
    return bFoundSpouse;
}

table Person tabCoupleAges2080
[calendar_year == 2080 && sex==FEMALE]
{
    has_partner *
    {
        duration()
    }
    *integer_age+
    *integer_age_partner+
};

table Person tabCoupleAges2140
[calendar_year == 2140 && sex==FEMALE]
{
    has_partner *
    {
        duration()
    }
    *integer_age+
    *integer_age_partner+
};


table Person tabCoupleAges2140MALE
[calendar_year == 2140 && sex==MALE]
{
    has_partner *
    {
        duration()
    }
    *integer_age+
    *integer_age_partner+
};

//EN UNION STATUS
table Person TabTestPartnershipStatus
{
    sex+ *
    {
        duration(has_partner, TRUE) / duration() //EN Proportion in partnership decimals=4
    }
    *integer_age +
    *calendar_year
};

//EN Partner Education
table Person TabTestPartnershipEduc2080
[ sex == FEMALE && has_partner && calendar_year == 2080 ]
{
    educ_fate+ *
    {
```

```
        duration(educ_partner, EL3_LOW) / duration(), //EN Low decimals=4
        duration(educ_partner, EL3_MEDIUM) / duration(), //EN Medium decimals=4
        duration(educ_partner, EL3_HIGH) / duration() //EN High decimals=4
    }
    * integer_age +
};

//EN Male Education
table Person TabTestMaleEduc2080
[ calendar_year == 2080 ]
{
    sex+ *
    {
        duration(educ_fate, EL3_LOW) / duration(), //EN Low decimals=4
        duration(educ_fate, EL3_MEDIUM) / duration(), //EN Medium decimals=4
        duration(educ_fate, EL3_HIGH) / duration() //EN High decimals=4
    }
    * integer_age +
};


//EN Male FATHER TYPE
table Person TabTestMaleFATHER2140
[ calendar_year == 2140 && sex==MALE]
{
    educ_fate+ *
    {
        duration(father_type, FT_NONO) / duration(), //EN NoNo decimals=4
        duration(father_type, FT_NOYES) / duration(), //EN NoYes decimals=4
        duration(father_type, FT_YESNO) / duration(), //EN YesNo decimals=4
        duration(father_type, FT_YESYES) / duration()  //EN YesYes decimals=4
    }
    * integer_age +
};
```

Step 13: Family Links

Overview

At this step we re-organize, revise, and maintain all family links by introducing a new dedicated module FamilyLinks.mpp. Family links are links between spouses, links between children and their biological parents, and of children with their parents or guardians with whom they currently live in a household. Family links are first established at birth and maintained at union formation, union dissolution, when leaving home, and at death. At each event the family status is updated (i.e. being a household head, spouse, or child) and all links are updated. For leaving home we create a new event. The update of links includes the children's decision with whom to stay after a union dissolution and the search for a suitable guardian if a lone parent dies.

The new FamilyLinks.mpp Module

This module handles and maintains family links. Family links are links between spouses, links between children and their biological parents, and of children with their parents or guardians with whom they currently live in a household. We model nuclear family households, consisting of a household head, and - if present - a spouse and dependent children.

Dependent children are:

- Children below age 18 who are not in a union or parents themselves
- Children below age 26 as long as they are continuously enrolled in education and not in a union or parents themselves - and who have not left home

Maintenance of links at death is handled in the function MaintainLinksAtDeath() linked to the mortality event. A spouse becomes the household head. If there is no spouse but children in the household, each child checks if she has a biological mother or father, or a grand mother or grand father alive, in which case she moves to a new guardian and updates all links. If not, and at age 18+, she becomes a household head.

Maintenance of links at union formations is handled in the function MaintainLinksAtUnionFormation() called at union formation. Partners become a household head and spouse and all children who lived with any of the two partners before in a household now update their family links as they now live with two "guardians". (The link to biological parents is kept over life and can be different from the links to the household head

and the spouse of the household head). If a union formation happens in the first year after giving birth and the baby has no biological father assigned, it is assumed that the male partner is the biological father of the baby.

Maintenance of links at union dissolution is handled in the function DissolveUnion() which is called when a union is dissolved. Both former partners now become household heads and children have to choose with whom to live. The choice is modeled by a set of simple rules and a probability to stay with the famale guardian. If only one of the two guardians is a biological parent, children choose to stay with the biological parent. Otherwise, the choice is random dependent on the parameter. All children who randomly decide make the same choice.

Leaving home is modelled as an event LeavingHomeEvent() which is called immediately after any union formation or increase of parity, at age 18 if not enrolled in school, at the moment of leaving school after age 18, or at age 26. In addition, children in education age 18-25 can leave home at any point in time following a parameter of age-specific home-leaving rates of students. Once a person has left the household, she is not moving back. The family status changes from child to household head or to spouse.

The creation of links at birth is handled in the function LinkFamilyAtBirth() hooked to the SetAliveEvent(). For children in the starting population it is assumed that both parents - if present in the household - are biological parents.

```
////////////////////////////////////////////////////////////////////////////////////
//
// Family Links
////////////////////////////////////////////////////////////////////////////////////
//

link Person.lBioFather Person.mlBioFatherChildren[];    //EN Biological Father - Children
link Person.lBioMother Person.mlBioMotherChildren[];    //EN Biological Mother - Children

link Person.lHHFather Person.mlHHFatherChildren[];      //EN HH Father - Children
link Person.lHHMother Person.mlHHMotherChildren[];      //EN HH Mother - Children

link Person.lSpouse;                                    //EN Spouses

////////////////////////////////////////////////////////////////////////////////////
//
// Dimensions
////////////////////////////////////////////////////////////////////////////////////
//

classification FAM_ROLE                                 //EN Role in family
{
    FR_HEAD,                                            //EN Head
    FR_SPOUSE,                                          //EN Spouse of head
```

```
    FR_CHILD                                        //EN Child
};

range AGE_18_25 { 18, 25 };                         //EN Age

////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    double ProbStayWithMother;        //EN Probability to stay with mother after union dissolution
    double ProbLeaveHome[AGE_18_25];  //EN Probability to leave home (students)
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor States and Functions
////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    FAM_ROLE    family_role = { FR_HEAD };                  //EN Family Role

    void        LinkFamilyAtBirth();                        //EN Establish family links at birth
    hook        LinkFamilyAtBirth, SetAliveEvent;           //EN Called at set alive event

    event       timeLeavingHomeEvent, LeavingHomeEvent;     //EN Leaving Home

    void        MaintainLinksAtUnionFormation();            //EN Family links at union formation
    hook        MaintainLinksAtUnionFormation, FindPartner; //EN Update at end of FindPartner()

    void        MaintainLinksAtDeath();                     //EN Family links at death
    hook        MaintainLinksAtDeath, MortalityEvent;       //EN Update at Death

    void        DissolveUnion();                            //EN Union Dissolution

    //EN Children in Household
    short       children_in_household = (sex == FEMALE) ?
                    sum_over(mlHHMotherChildren, is_alive) :
                    sum_over(mlHHFatherChildren, is_alive);
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////
//


void Person::MaintainLinksAtDeath()
{
    if (lSpouse) lSpouse->family_role = FR_HEAD;
    else if (children_in_household > 0)
    {
        //try to find somebody for children
        int nIndex;
        auto prChild = (sex==FEMALE) ? mlHHMotherChildren->GetNext( 0, &nIndex ) :
mlHHFatherChildren->GetNext( 0, &nIndex );
        while ( prChild != NULL )
        {
            Person * prGuardian = NULL;
            if (sex == FEMALE && prChild->lBioFather)    // own biological father
```

```
            {
                prGuardian = prChild->lBioFather;
            }
            else if (sex == MALE && prChild->lBioMother)  // own biological mother
            {
                prGuardian = prChild->lBioMother;
            }
            else if (lBioMother)                          // grandmother
            {
                prGuardian = lBioMother;
            }
            else if (lBioFather)                          // grandfather
            {
                prGuardian = lBioFather;
            }
            else if (prChild->integer_age >= 18)          // nobody
            {
                prChild->family_role = FR_HEAD;
            }
            if (prGuardian && prGuardian->sex == MALE)
            {
                prChild->lHHFather = prGuardian;
                if (prGuardian->lSpouse) prChild->lHHMother = prGuardian->lSpouse;
            }
            else if (prGuardian && prGuardian->sex == FEMALE)
            {
                prChild->lHHMother = prGuardian;
                if (prGuardian->lSpouse) prChild->lHHFather = prGuardian->lSpouse;
            }
            prChild = (sex==FEMALE) ? mlHHMotherChildren->GetNext( nIndex+1, &nIndex ) :
mlHHFatherChildren->GetNext( nIndex+1, &nIndex );
        }
    }
}


void Person::DissolveUnion()
{
    if (lSpouse)
    {
        //For all children in household decide with whom to stay
        if (children_in_household)
        {
            int nIndex;
            bool bStayWithMother = (RandUniform(32) < ProbStayWithMother);
            //Stay with mother
            auto prChild = mlHHMotherChildren->GetNext( 0, &nIndex );
            while ( prChild != NULL )
            {
                if (bStayWithMother && ((prChild->lBioMother && prChild->lBioMother == this)
                    || !prChild->lBioFather || (prChild->lBioFather && prChild->lBioFather !=
lSpouse)))
                {
                    prChild->lHHFather = NULL;
                }
                prChild = mlHHMotherChildren->GetNext(nIndex + 1, &nIndex);
            }
            //Stay with father
            if (lSpouse->children_in_household)
            {
                auto prChild = lSpouse->mlHHFatherChildren->GetNext( 0, &nIndex );
                while ( prChild != NULL )
                {
                   prChild->lHHMother = NULL;
                   prChild = lSpouse->mlHHFatherChildren->GetNext(nIndex + 1, &nIndex);
                }
```

```
            }
        }
        lSpouse->family_role = FR_HEAD;
        family_role = FR_HEAD;
        lSpouse = NULL;
    }
}

void Person::MaintainLinksAtUnionFormation()
{
    // Partners children in HH now live also with this woman
    if (lSpouse && lSpouse->children_in_household)
    {
        int nIndex;
        auto prChild = lSpouse->mlHHFatherChildren->GetNext( 0, &nIndex );
        while ( prChild != NULL )
        {
            prChild->lHHMother = this;
            prChild = lSpouse->mlHHFatherChildren->GetNext(nIndex + 1, &nIndex);
        }
    }
    if (lSpouse)
    {
        // Own children in HH also live with partner now
        int nIndex;
        auto prChild = mlHHMotherChildren->GetNext( 0, &nIndex );
        while ( prChild != NULL )
        {
            prChild->lHHFather = lSpouse;
            // if babies without biological father, the spouse becomes biological father
            if (!prChild->lBioFather && prChild->integer_age == 0)
            {
                prChild->lBioFather = lSpouse;
            }
            prChild = mlHHMotherChildren->GetNext(nIndex + 1, &nIndex);
        }
        family_role = FR_HEAD;
        lSpouse->family_role = FR_SPOUSE;

        // own household
        lHHFather = NULL;
        lHHMother = NULL;
        lSpouse->lHHFather = NULL;
        lSpouse->lHHMother = NULL;
    }
}

TIME Person::timeLeavingHomeEvent()
{
    if (in_projected_time && family_role == FR_CHILD
        && (integer_age >= 26 || lSpouse || (integer_age >= 18 && !in_school) || parity > 0))
    {
        return WAIT(0);
    }
    else if (family_role == FR_CHILD && WITHIN(AGE_18_25, integer_age))
    {
        double dProb = ProbLeaveHome[RANGE_POS(AGE_18_25, integer_age)];
        if (dProb >= 1.0) return WAIT(0);
        else if (dProb > 0.0) return WAIT(-log(RandUniform(33)) / -log(1 - dProb));
        else return TIME_INFINITE;
    }
    else return TIME_INFINITE;
}

void Person::LeavingHomeEvent()
{
```

```
    // become a head or a spouse
    if (lSpouse && lSpouse->family_role == FR_HEAD) family_role = FR_SPOUSE;
    else family_role = FR_HEAD;

    // remove links to household
    lHHFather = NULL;
    lHHMother = NULL;
}

void Person::LinkFamilyAtBirth()
{
    if (person_type == PT_START && family_role != FR_HEAD && peHHead != NULL)
    {
        if (family_role == FR_CHILD)
        {
            if (peHHead->sex == MALE)
            {
                lBioFather = peHHead;
                if (lBioFather->lSpouse) lBioMother = lBioFather->lSpouse;
            }
            else
            {
                lBioMother = peHHead;
                if (lBioMother->lSpouse) lBioFather = lBioMother->lSpouse;
            }
            lHHFather = lBioFather;
            lHHMother = lBioMother;
        }
        else
        {
            lSpouse = peHHead;
            if (sex == lSpouse->sex) // FIX: do not allow same sex couples
            {
                if (sex == FEMALE) sex = MALE; else sex = FEMALE;
            }
        }
    }
    else if (person_type == PT_CHILD)
    {
        lBioMother = peHHead;
        if (lBioMother->lSpouse) lBioFather = lBioMother->lSpouse;
        lHHFather = lBioFather;
        lHHMother = lBioMother;
    }
}


////////////////////////////////////////////////////////////////////////////////////////////
//
// Validation Tables
////////////////////////////////////////////////////////////////////////////////////////////
//


table Person TabFamilyRoleTest //EN FAMILY ROLE
[WITHIN(SIM_YEAR_RANGE, calendar_year)]
{
    sex+ *
    {
        duration(family_role, FR_CHILD) / duration(), //EN Child
        duration(family_role, FR_HEAD) / duration(), //EN Head
        duration(family_role, FR_SPOUSE) / duration() //EN Spouse
    }
    * integer_age
    * sim_year
};
```

Step 14: Education Alignment

Overview

At this step we add a module EducationAlignment.mpp which allows aligning school attendance to external school enrolment parameters. The module is optional and can be switched off by the user.

The new EducationAlignemnt.mpp Module

This module allows aligning school attendance to external school enrolment parameters. This module is optional and can be switched off by the user. The module works on top of the education pattern module and does not interfere with the school attendance patterns produced there. Alignment of education rates works in one direction only - namely increasing the number of students. This is to add studies which are not essential to reach the highest grade and to reproduce external enrolment targets. If the expected number of students by age and sex is higher in the alignment targets than produced by the pattern model, additional people are flagged as students by setting the state in_other_education to TRUE. The second state affected is in_school which can be true due to 'normal' school attendance (in_regular_school) or due to the additional attendance.

The algorithm is currently kept very simple: eligible for becoming a student in_other_education are those who have been in this status the school year before or those just finishing regular education. A second criterion is family: those living with children are excluded from the pool. People available for additional studies are defined in the actor set asAvailableForAdditionalStudies which can be easily modified to add realism. Alignments are made by the function AlignAfterSchoolYearChange(). The function does not force alignment if no additional students are found. This could be changed by increasing the pool of potential students, e.g. allowing people resuming studies after having left school before, or by allowing people with parenting obligations continuing or resuming studies.

```
////////////////////////////////////////////////////////////////////////////////////////
//
// Actor sets
////////////////////////////////////////////////////////////////////////////////////////
//

//EN People eligible for additional studies after finishing school
actor_set Person asAvailableForAdditionalStudies[sex][integer_age]
    filter (year_finish_school == calendar_year || in_other_education)
    && children_in_household == 0 && !is_flagged_enrol_other_school;
```

```
//EN All Persons by age and sex
actor_set Person asAllPersonsSexAge[sex][integer_age] filter is_alive;

//EN All in regular education
actor_set Person asAllInRegularSchool[sex][integer_age] filter in_regular_school;

range AGE_EDUC_ALIGN { 15,30 }; //EN Age


////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN School Enrolment rates (for alignment)
    double SchoolEnrolmentRates[SEX][AGE_EDUC_ALIGN][SIM_YEAR_RANGE];

    //EN Align school enrolment rates
    logical AlignSchoolEnrolmentRates;
};

parameter_group PG_EducAlignment    //EN Education Alignment
{
  SchoolEnrolmentRates, AlignSchoolEnrolmentRates
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states and functions
////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    //EN Person gegularely enrolled in school
    logical  in_regular_school = (educ_status == ES_FULLTIME || educ_status == ES_PARTTIME
            || educ_status == ES_DUAL) ? TRUE : FALSE;

    //EN Person in other education (used to align in_school rates to higher targets)
    logical in_other_education = { FALSE };

    //EN Person enrolled in school incl. other (alignment)
    logical  in_school = (in_regular_school || in_other_education) ? TRUE : FALSE;

    //EN Person flagged for enrolling in other education
    logical is_flagged_enrol_other_school = { FALSE };
};

actor Clock
{
    void AlignAfterSchoolYearChange();                  //EN Align school attendance
    hook AlignAfterSchoolYearChange, SchoolYearClock;   //EN Hook school alignment to clock

};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////
//


void Clock::AlignAfterSchoolYearChange()
```

```
{
    if (AlignSchoolEnrolmentRates)
    {
        for (int nAge = MIN(AGE_EDUC_ALIGN); nAge <= MAX(AGE_EDUC_ALIGN); nAge++)
        {
            for (int nSex = 0; nSex < SIZE(SEX); nSex++)
            {
                long nPopAvailable = asAvailableForAdditionalStudies[nSex][nAge]->Count();
                long nPop = asAllPersonsSexAge[nSex][nAge]->Count();
                long nAllInRegularSchool = asAllInRegularSchool[nSex][nAge]->Count();

                long nMissing = long(nPop * SchoolEnrolmentRates[nSex]
                    [RANGE_POS(AGE_EDUC_ALIGN, nAge)][RANGE_POS(SIM_YEAR_RANGE, clock_year)]
                    - nAllInRegularSchool);

                while (nMissing > 0 && asAvailableForAdditionalStudies[nSex][nAge]->Count() > 0)
                {
                    auto prNewStudent = asAvailableForAdditionalStudies[nSex][nAge]
                        ->GetRandom(RandUniform(34));
                    prNewStudent->is_flagged_enrol_other_school = TRUE;
                    nMissing--;
                }
                for (long nIndex = 0; nIndex < nPop; nIndex++)
                {
                    auto prPerson = asAllPersonsSexAge[nSex][nAge]->Item(nIndex);
                    if (prPerson->is_flagged_enrol_other_school) prPerson->in_other_education =
TRUE;
                    else  prPerson->in_other_education = FALSE;
                    prPerson->is_flagged_enrol_other_school = FALSE;
                }
            }
        }
        // kick those too old out of school
        long nPop = asAllPersonsSexAge[FEMALE][MAX(AGE_EDUC_ALIGN)+1]->Count();
        for (long nIndex = 0; nIndex < nPop; nIndex++)
        {
            asAllPersonsSexAge[FEMALE][MAX(AGE_EDUC_ALIGN) + 1]
                ->Item(nIndex)->in_other_education = FALSE;
        }
        nPop = asAllPersonsSexAge[MALE][MAX(AGE_EDUC_ALIGN)+1]->Count();
        for (long nIndex = 0; nIndex < nPop; nIndex++)
        {
            asAllPersonsSexAge[MALE][MAX(AGE_EDUC_ALIGN) + 1]->Item(nIndex)
                ->in_other_education = FALSE;
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////
//
// Validation
////////////////////////////////////////////////////////////////////////////////////////////
//

table Person TabEducAlignmentValidation     //EN Education Alignment
{
    {
        duration(in_school, TRUE) / duration(),           //EN In any school decimals=4
        duration(in_regular_school, TRUE) / duration(),   //EN In regular school decimals=4
        duration(in_other_education, TRUE) / duration()   //EN In other school decimals=4
    }
    * integer_age
    * calendar_year
};
```

Step 15: Emigration

Overview

At this step we add an emigration module. Emigration is optional and can be deactivated. Users have various choices in how emigration is modeled. In order to allow for alignment, emigration is decided on an annual basis each mid-year.

The new Emigration.mpp Module

This module implements emigration. Users can choose three approaches:

- Emigration based on age and sex specific emigration rates: Individual emigration based on time-invariant rates
- Emigration based on a parameter of total emigration: Individual emigration based on age and sex specific rates proportionally adjusted to meet aggregate totals
- Family emigration: This option keeps families together and meets the overall total emigration targets. In the current implementation, the person with the shortest waiting time is chosen and takes the whole family abroad until the total target is met. This approach does not maintain the age-specific rates and might be further developed to better fill a given age-distribution of emigrants

The module is prepared to include individual-level differences in emigration rates (e.g. by nationality) and automatically adjusts individual rates in order to meet aggregate targets. Emigration is decided each mid-year and happens only annualy at this point of time. Emigrants leave the country and are immediately removed from the simulation.

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor sets
////////////////////////////////////////////////////////////////////////////////////////////
//

//EN Residents by sex and age group
actor_set Person asResidentSexAge5[sex][age5_index]
    filter is_alive && is_resident && in_projected_time;

//EN Residents by sex and age group - ordered by waiting time
actor_set Person asResidentOrderedSexAge5[sex][age5_index]
    filter is_alive && is_resident && in_projected_time
    order emigr_wait;

//EN Residents by sex - ordered by waiting time
actor_set Person asResidentOrdered
    filter is_alive && is_resident && in_projected_time
```

```
    order emigr_wait;

//EN Entire resident population
actor_set Person asAllResidents filter is_alive && is_resident;

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////////////
//


partition AGE5_PART                                         //EN Age Groups
{
    5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,
    65, 70, 75, 80, 85, 90
};

range AGE5_INDEX { 0,18 };                                  //EN Age group

classification EMIGR_SET                                    //EN Emigration settings
{
    ES_AGERATES,                                            //EN Meet rates by age group and sex
    ES_TOTALS,                                              //EN Meet aggregate totals
        ES_FAMILY                                                       //EN Family
Emigration
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{

    logical     ModelEmigration;                            //EN Switch emigration on/off
    EMIGR_SET   EmigrationSettings;                         //EN Emigration targets
    double      EmigrationRates[SEX][AGE5_PART];            //EN Emigration rates by sex and age
    long        EmigrationTotal[SIM_YEAR_RANGE];            //EN Total number of emigrants
};

parameter_group PG_EmigrationBase                           //EN Emigration Base Version
{
    ModelEmigration, EmigrationSettings,
    EmigrationRates, EmigrationTotal
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states and Events
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    SIM_YEAR_RANGE sim_year = COERCE(SIM_YEAR_RANGE, calendar_year);    //EN Year
    AGE5_INDEX  age5_index = split(integer_age, AGE5_PART);     //EN Age Group
    logical     is_resident = { TRUE };                         //EN Is resident
    logical     has_emigrated = { FALSE };                      //EN Person has emigrated
    double      emigr_prob = { 0.0 };                           //EN Emigration Probabilty
    void        DoEmigrate();                                   //EN Do emigrate
    TIME        emigr_wait = { TIME_INFINITE };                 //EN Hypothetical waiting time
    integer     DoEmigrateChildren();                           //EN Send children abroad
};
```

```
actor Clock
{
    void        DoEmigration();                                 //EN Do Emigration
    hook        DoEmigration, MidYearClockEvent, 5;               //EN hook to mid year
    double      AdjustedProb(double dProb, double dFactor);     //EN Adjust Probability
    double      ProbFromOdds(double dOdds);                     //EN Get Probability from Odds
    double      OddsFromProb(double dProb);                     //EN Get Odds from Probability
};

actor Globals                                                   //EN Actor Globals
{
    //EN Emigration alignment factors
    double      emigration_alignment[SIM_YEAR_RANGE][SEX][AGE5_PART];
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////
//

void Clock::DoEmigration()
{
    if (clock_year >= MIN(SIM_YEAR_RANGE) && ModelEmigration == TRUE)
    {
        double dExpectedEmigrants = 0.0;
        long   nExpectedEmigrants = 0;
        double dExpectedEmigrationProb = 0.0;
        double dCurrentEmigrationProb = 0.0;
        long   nGroupSize = 0;
        // for each age group by sex
        // find, record and apply an alignment factor to meet target group emigration rates
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            for (int nGroup = 0; nGroup < SIZE(AGE5_INDEX); nGroup++)
            {
                // The expected emigration pribability in this group
                dExpectedEmigrationProb = EmigrationRates[nSex][nGroup];

                // Update the individual emigration probabilities of all members of this group
                dCurrentEmigrationProb = 0.0;
                nGroupSize = asResidentSexAge5[nSex][nGroup]->Count();
                for (long nIndex = 0; nIndex < nGroupSize; nIndex++)
                {
                    auto prPerson = asResidentSexAge5[nSex][nGroup]->Item(nIndex);
                    double dPersonalEmiProb = 0.0;
                    // Calculate the current personal probability to emigrate

                    //
//////////////////////////////////////////////////////////////////////////
                    // At this place, individual level odds can be introduced do adjust individual
rates
                    //
==========================================================================
                    /* dPersonalEmiProb = ProbFromOdds(OddsFromProb(EmigrationRates[prPerson-
>sex][prPerson->age5_index])
                        * EmigrationOdds[prPerson->sex][prPerson->nation_agg][prPerson-
>age5_index] ); */
                    //
//////////////////////////////////////////////////////////////////////////

                    dPersonalEmiProb = EmigrationRates[prPerson->sex][prPerson->age5_index];
                    prPerson->emigr_prob = dPersonalEmiProb;
                    dCurrentEmigrationProb = dCurrentEmigrationProb + dPersonalEmiProb;
                }
```

```
                // This is the current overall probability in the group
                if (nGroupSize > 0) dCurrentEmigrationProb = dCurrentEmigrationProb / nGroupSize;

                // Find alignment factor for individual emigration probabilities
                double dLower = 0.0;
                double dUpper = 200.0;
                double dCenter = 100.0;
                int nMaxIter = 1000000;
                bool bEnterAlignment = FALSE;
                while (abs(dCurrentEmigrationProb - dExpectedEmigrationProb) > 0.00001 && nMaxIter
> 0 && nGroupSize > 0)
                {
                    bEnterAlignment = TRUE;
                    dCurrentEmigrationProb = 0.0;
                    dCenter = (dLower + dUpper) / 2.0;
                    for (long nIndex = 0; nIndex < nGroupSize; nIndex++)
                    {
                        auto prPerson = asResidentSexAge5[nSex][nGroup]->Item(nIndex);
                        dCurrentEmigrationProb = dCurrentEmigrationProb + AdjustedProb(prPerson-
>emigr_prob, dCenter);
                    }
                    dCurrentEmigrationProb = dCurrentEmigrationProb / nGroupSize;
                    nMaxIter--;
                    if (dCurrentEmigrationProb > dExpectedEmigrationProb) dUpper = dCenter;
                    else dLower = dCenter;
                }
                // Record it
                if (bEnterAlignment) asGlobals->Item(0)-
>emigration_alignment[RANGE_POS(SIM_YEAR_RANGE, clock_year)][nSex][nGroup] = dCenter;
                else asGlobals->Item(0)->emigration_alignment[RANGE_POS(SIM_YEAR_RANGE,
clock_year)][nSex][nGroup] = 1.0;
                // Apply it to update all individual emigration probabilities
                if (bEnterAlignment)
                {
                    for (long nIndex = 0; nIndex < nGroupSize; nIndex++)
                    {
                        auto prPerson = asResidentSexAge5[nSex][nGroup]->Item(nIndex);
                        prPerson->emigr_prob = AdjustedProb(prPerson->emigr_prob, dCenter);
                    }
                }
                // set hypothetical waiting times
                for (long nIndex = 0; nIndex < nGroupSize; nIndex++)
                {
                    auto prPerson = asResidentSexAge5[nSex][nGroup]->Item(nIndex);
                    double dProb = prPerson->emigr_prob;
                    if (dProb == 1) prPerson->emigr_wait = 0.0;
                    else if (dProb == 0) prPerson->emigr_wait = TIME_INFINITE;
                    else prPerson->emigr_wait = -log(RandUniform(20)) / -log(1 - dProb);
                }
            }
        }

        // Make people emigrate
        // The model used can be chosen by user
        // (1) Option 1: Individual Emigration meeting rates by age group and sex

        if (EmigrationSettings == ES_AGERATES)
        {
            // for each age group by sex
            for (int nSex = 0; nSex < SIZE(SEX); nSex++)
            {
                for (int nGroup = 0; nGroup < SIZE(AGE5_INDEX); nGroup++)
                {
                    // integer number of emigrants in group
                    dExpectedEmigrants = EmigrationRates[nSex][nGroup] *
(asResidentSexAge5[nSex][nGroup]->Count());
```

```
                    nExpectedEmigrants = int(dExpectedEmigrants);
                    if (RandUniform(21) < dExpectedEmigrants - (double)nExpectedEmigrants)
nExpectedEmigrants++;

                    // Call emigration for persons with shortest waiting time
                    for (int nIndex = 0; nIndex < nExpectedEmigrants; nIndex++)
                    {
                        auto prPerson = asResidentOrderedSexAge5[nSex][nGroup]->Item(0);
                        prPerson->DoEmigrate();
                    }
                }
            }
        }
                // (2) Option 2: Individual emigration meeting aggregate yearly targets
                else if (EmigrationSettings == ES_TOTALS)
                {
                        // integer number of emigrants in group
                        dExpectedEmigrants = (double)EmigrationTotal[RANGE_POS(SIM_YEAR_RANGE,
clock_year)] / asGlobals->Item(0)->person_weight;
                        nExpectedEmigrants = int(dExpectedEmigrants);
                        if (RandUniform(23) < dExpectedEmigrants - (double)nExpectedEmigrants)
nExpectedEmigrants++;

                        // Call emigration for persons with shortest waiting time
                        for (int nIndex = 0; nIndex < nExpectedEmigrants; nIndex++)
                        {
                                auto prPerson = asResidentOrdered->Item(0);
                                prPerson->DoEmigrate();
                        }
                }
                // (3) Option 3: Family Emigration
                else if (EmigrationSettings == ES_FAMILY)
                {
                        // integer number of emigrants in group
                        dExpectedEmigrants = (double)EmigrationTotal[RANGE_POS(SIM_YEAR_RANGE,
clock_year)] / asGlobals->Item(0)->person_weight;
                        nExpectedEmigrants = int(dExpectedEmigrants);
                        if (RandUniform(35) < dExpectedEmigrants - (double)nExpectedEmigrants)
nExpectedEmigrants++;

                        // Call emigration for persons with shortest waiting time
                        int nIndex = 0;
                        while ( nIndex < nExpectedEmigrants )
                        {
                                auto prPerson = asResidentOrdered->Item(0);
                                // find family head
                                if (prPerson->family_role == FR_SPOUSE && prPerson->lSpouse)
prPerson = prPerson->lSpouse;
                                else if (prPerson->family_role == FR_CHILD)
                                {
                                        if (prPerson->lHHFather && prPerson->lHHFather-
>family_role == FR_HEAD) prPerson = prPerson->lHHFather;
                                        else if (prPerson->lHHMother && prPerson->lHHMother-
>family_role == FR_HEAD) prPerson = prPerson->lHHMother;
                                }
                // send children abroad
                if (prPerson->family_role != FR_CHILD)
                {
                   nIndex = nIndex + prPerson->DoEmigrateChildren();
                }
                // send spouse abroad
                                if (prPerson->lSpouse)
                                {
                                        prPerson->lSpouse->DoEmigrate();
                                        nIndex++;
                                }
```

```
                // leave country
                prPerson->DoEmigrate();
                                nIndex++;
                        }
                }
    }
}

integer Person::DoEmigrateChildren()
{
    int ReturnValue = 0;
    int nChildIndex;
    auto prChild = (sex == FEMALE) ? mlHHMotherChildren->GetNext(0, &nChildIndex) :
mlHHFatherChildren->GetNext(0, &nChildIndex);
        while (prChild != NULL)
        {
            prChild->DoEmigrate();
        ReturnValue++;
                prChild = (sex == FEMALE) ? mlHHMotherChildren->GetNext(nChildIndex + 1,
&nChildIndex) : mlHHFatherChildren->GetNext(nChildIndex + 1, &nChildIndex);
        }
    return      ReturnValue;
}

double Clock::AdjustedProb(double dProb, double dFactor)
{
    return ProbFromOdds(OddsFromProb(dProb) * dFactor);
}

// Get Probability from Odds
double Clock::ProbFromOdds(double dOdds)
{
    return dOdds / (1 + dOdds);
}

// Get Odds from Probability
double Clock::OddsFromProb(double dProb)
{
    if (dProb < 1) return dProb / (1 - dProb);
    else return TIME_INFINITE;
}

void Person::DoEmigrate()
{
    has_emigrated = TRUE;
    is_resident = FALSE;
    Finish();
}

////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////
//

table Person TabEmigration                                      //EN Emigration Rates
[is_alive && in_projected_time]
{
    sex + *
    {
        transitions(is_resident, TRUE, FALSE) / duration()    //EN Emigration rates decimals=4
    }
    * split(integer_age, AGE5_PART)
    * sim_year
};
```

```
table Person TabEmigration2                                          //EN Emigration Numbers
[is_alive && in_projected_time]
{
    sex + *
    {
        transitions(is_resident, TRUE, FALSE)                        //EN Emigration numbers
    }
    * split(integer_age, AGE5_PART) +
    * sim_year
};

table_group TG_Emigration        //EN Emigration tables
{
    TabEmigration, TabEmigration2
};
```

Step 16: Immigration

Overview

At this step we add an immigration module. Immigration is optional and can be deactivated. Immigrant children try to find an appropriate mother destined to immigrate in the same calendar year. Before entering the country, immigrants are excluded from many processes like education, fertility or partnership formation. Missing characteristics are cloned from a randomly chosen resident of the same age and sex at the moment of entry.

The new Immigration.mpp Module

This module implements immigration. It is parameterized by the total number of immigrants and the age-sex distribution of immigrants. Children younger than 18 try to identify a mother in the population of women destined to immigrate in the same year. While the immigration module is very simple concerning scenario creation allowing to easily reproduce macro projections, it requires adaptation of code in various other modules:

- Most tables and actor-sets throughout the model were adapted to only include the resident population (is_resident)
- Various processes, e.g. education, require additional filters to only include residents.
- The Start() function has to be adapted to take the time of immigration as additional parameter - and to initialize states of immigrants.
- The SetAliveEvent() calls a function for children to identify and link to a mother immigrating in the same year
- In the Simulation() function, immigrants have to be created

Various characteristis including a set of education states are sampled from residents of the same age and sex at the moment of immigration in the ImmigrationEvent(). In this model implementation, immigrants look like residents concerning the distributions of most characteristics.

```
////////////////////////////////////////////////////////////////////////////////////
//
// Actor sets
////////////////////////////////////////////////////////////////////////////////////
//

//EN Potential immigrant mothers
actor_set Person asPotentialImmigrantMothers[integer_age][year_of_immigration]
    filter is_alive && sex==FEMALE && parity < 6 && person_type == PT_IMMIGRANT;
```

```
//EN All Residents
actor_set Person asAllResidents filter is_alive && is_resident;  //EN Entire resident population

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    logical     ModelImmigration;                           //EN Switch immigration on/off
    long        ImmigrationTotal[SIM_YEAR_RANGE];           //EN Total number of immigrants
    cumrate [2] ImmigrationAgeSexAll[SEX][AGE_RANGE];       //EN Age-Sex distribution of
immigrants
    cumrate [1] AgeOfImmigrantMother[FERTILE_AGE_RANGE];    //EN Age distribution of mothers at
birth
};

parameter_group PG_ImmigrationBase                          //EN Immigration Base Version
{
    ModelImmigration, ImmigrationTotal,
    ImmigrationAgeSexAll, AgeOfImmigrantMother
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states and functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
   //EN Immigrant
    logical         is_immigrant = (person_type == PT_IMMIGRANT) ? TRUE : FALSE;

    SIM_YEAR_RANGE  year_of_immigration = { 2050 };         //EN Year of immigration
    TIME            time_of_immigration = {TIME_INFINITE};  //EN Time of immigration
    event           timeImmigrationEvent, ImmigrationEvent; //EN Immigration Event
    logical         FindImmigrantMother();                  //EN Find a mother immigrating same
year
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

TIME Person::timeImmigrationEvent()
{
    if (!ever_resident) return time_of_immigration;
    else return TIME_INFINITE;
}

logical Person::FindImmigrantMother()
{
    bool bFoundMother = FALSE;
    double dAgeImmi = time_of_immigration - time_of_birth;
    if (dAgeImmi < 18.0)
    {
        int nAge;
        int nCount = 0;
        bool bFound = FALSE;
        while (!bFound && nCount < 100)
        {
```

```
            Lookup_AgeOfImmigrantMother(RandUniform(39), &nAge);
            nAge = nAge + MIN(FERTILE_AGE_RANGE);
            if (asPotentialImmigrantMothers[nAge][RANGE_POS(SIM_YEAR_RANGE, year_of_immigration)]-
>Count() > 0) bFound = TRUE;
            nCount++;
        }
        if (asPotentialImmigrantMothers[nAge][RANGE_POS(SIM_YEAR_RANGE, year_of_immigration)]-
>Count() > 0)
        {
            auto prMother = asPotentialImmigrantMothers[nAge][RANGE_POS(SIM_YEAR_RANGE,
year_of_immigration)]->Item(RandUniform(40));
            lBioMother = prMother;
            lHHMother =  prMother;
            prMother->parity = prMother->parity + 1;
            prMother->is_mother = TRUE;
            bFoundMother = TRUE;
        }
    }
    return bFoundMother;
}

void Person::ImmigrationEvent()
{
    // Find a resident host of same sex and age and clone characteristics
    if (asAllPersonsSexAge[sex][integer_age]->Count() > 0)
    {
        auto prHostPerson = asAllPersonsSexAge[sex][integer_age]->GetRandom(RandUniform(45));

        // Clone education states from host
        educ_level              = prHostPerson->educ_level;
        educ_status             = prHostPerson->educ_status;
        educ_pattern_status     = prHostPerson->educ_pattern_status;
        in_other_education      = prHostPerson->in_other_education;
        year_finish_school      = prHostPerson->year_finish_school;
        educ_pattern_number     = prHostPerson->educ_pattern_number;
        educ_fate               = prHostPerson->educ_fate;
        educ_prob1              = prHostPerson->educ_prob1;
        educ_prob2              = prHostPerson->educ_prob2;
        educ_parents            = prHostPerson->educ_parents;
        educ_parents_is_known   = prHostPerson->educ_parents_is_known;

        // Cloneo other characteristics from host
        never_father            = prHostPerson->never_father;
    }
    // Residence status
    is_resident = TRUE;
    ever_resident = TRUE;
}

////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////
//

table Person tabNumberImmigrants                        //EN Number of Immigrants
[is_alive && in_projected_time]
{
    sex + *
    {
        transitions(ever_resident,FALSE,TRUE)           //EN Number Immigrants
    }
    * integer_age+
    * sim_year
};
```

```
table Person tabRoleValidTab                              //EN Validation Family Role by Age
[is_resident && in_projected_time && is_resident]
{
    person_type+ *
    {
        duration(family_role,FR_CHILD) / duration(),    //EN Child decimals=4
        duration(family_role,FR_SPOUSE) / duration(),   //EN Spouse decimals=4
        duration(family_role,FR_HEAD) / duration()      //EN Head decimals=4
    }
    * integer_age
    * sim_year
};
```

Changes in the Simulation() function

```
    // Create Immigrants
        if (ModelImmigration)
        {
                for (int nYear = MIN(SIM_YEAR_RANGE); nYear < int(SIMULATION_END()); nYear++)
                {
                        int nCohortSize = int(ImmigrationTotal[RANGE_POS(SIM_YEAR_RANGE, nYear)] /
(asGlobals->Item(0)->person_weight));
                        double dCohortSize = ImmigrationTotal[RANGE_POS(SIM_YEAR_RANGE, nYear)] /
(asGlobals->Item(0)->person_weight);
                        if (RandUniform(42) < dCohortSize - nCohortSize) nCohortSize++;
                        for (int nIndex = 0; nIndex < nCohortSize; nIndex++)
                        {
                                auto prPerson = new Person();
                                prPerson->Start(NULL, NULL, RandUniform(43) + nYear);
                        }
                }
        }
```

Changes in the PersonCore.mpp module

Changes affect the Start() function and the SetAliveEvent()

```
void Person::Start(Observation *peObservation, Person *pePers,  TIME dTimeOfImmigration)
{
    // Setting the actor weight
    Set_actor_weight(asGlobals->Item(0)->person_weight);
    Set_actor_subsample_weight(asGlobals->Item(0)->person_weight);

    // Determine the person type
    if (peObservation == NULL && pePers != NULL ) person_type = PT_CHILD;   // Born in simulation
    else if (peObservation != NULL ) person_type = PT_START;                // From Starting Pop
    else person_type = PT_IMMIGRANT;                                        // Immigrant

    // Initialize states
    if (person_type == PT_START) // Person comes from starting population
    {
        // (A) States from Starting population file
        time                = peObservation->pmc[PMC_BIRTH] + RandUniform(2);
        sex                 = (SEX)(int)peObservation->pmc[PMC_SEX];
```

```
        family_role         = (FAM_ROLE)( int )peObservation->pmc[PMC_ROLE];
        educ_startpop       = (EDUC_LEVEL3)(int)peObservation->pmc[PMC_EDUC];
        in_school_startpop  = (bool)(int)peObservation->pmc[PMC_INSCHOOL];

        // (B) Other states
        time_of_birth       = time;
        calendar_year       = (int)time_of_birth;

        // (C) Links to head resp. spouse
        if (pePers) peHHead = pePers;
    }

    else if (person_type == PT_CHILD) // Person born in simulation
    {
        // (A) States corresponding to Starting population file variables
        if (RandUniform(4) < 100.0 / (100.0 + SexRatio[RANGE_POS(SIM_YEAR_RANGE, calendar_year)]))
        {
            sex = FEMALE;
        }
        else sex = MALE;
        time = pePers->time;
        family_role = FR_CHILD;

        // (B) Other states
        time_of_birth = time;
        calendar_year = ( int )time_of_birth;

        // (C) Links to head resp. spouse
        peHHead = pePers;
    }

    else // Person is an immigrant
    {
                is_resident = FALSE;
                ever_resident = FALSE;
                time_of_immigration = dTimeOfImmigration;
                year_of_immigration = (int)dTimeOfImmigration;
                family_role = FR_HEAD;

                int nSex, nAge, nNation;
                Lookup_ImmigrationAgeSexAll(RandUniform(41),&nSex, &nAge);
                sex = (SEX)nSex;
                time = dTimeOfImmigration - nAge - RandUniform(44);

                // (B) Other states
                time_of_birth = time;
                calendar_year = (int)time_of_birth;
    }
    time_set_alive = WAIT(0);
}

/*  NOTE(Person.SetAliveEvent, EN)
    This function is called with waiting time 0 immediately after the Start of a Person actor.
    For people of the starting population the date of birth is only known after getting this
    information from the corresponding person record, thus after the person actor is created.
    The SetAliveEvent Event ensures that no person is visible and counted before its actual
    birth date.
*/

TIME Person::timeSetAliveEvent() { return time_set_alive; }
void Person::SetAliveEvent()
{
    lClock          = asTheClock->Item(0);      // link the person to the clock
    lGlobals        = asGlobals->Item(0);       // link the person to globals

    is_alive        = TRUE;                     // set the Person alive
```

```
    time_set_alive = TIME_INFINITE;             // ensure the event does not happen again

    IMPLEMENT_HOOK();

    if (person_type == PT_IMMIGRANT)
    {
        if (FindImmigrantMother()) family_role = FR_CHILD;
    }

    peHHead = NULL;
}
```

Step 17: Basic NTA

Overview

At this step we add a new module for NTA variables by sex, education and family type. According to the current life-course situation, a list of NTA variables with the base year's NTA values is maintained. In order to allow comparing published with disagrregated NTA variables within a single simulation - and to add user choices concerning the disaggregation level of NTA data used in a simulation - we include paramers for published NTA variables and model selection. Users can choose to use (1) NTA by age, (2) NTA by age and sex, or (3) NTA by age, sex, family type and education.

The new NtaBase.mpp Module

This module implements the base NTA parameters and variables by sex, education and family type. For children and students, education refers to parent's education, for all others the own educational attainment. By age and school enrolment we distinguish 6 person types:

- Children 0-16
- Students 17-25
- Non-students 17-25
- Adults 26-59
- Adults 60+

For the non-student population 17-25 and adults 26-59, we distinguish 6 family types:

- Single not living with children age <25
- Single living with young child(ren) age < 3
- Single living with older child(ren) < 25
- Couple not living with children age <25
- Couple living with young child(ren) age < 3
- Couple living with older child(ren) < 25

For the population 60+ we distinguis 4 family types:

- Single childless
- Single ever parent
- Couple childless

- Couple ever parent

The module maintains the individual base NTA variables depending on the current individual characteristics from the according parameters. The list of variables is:

- Private Consumption Education (CFE)
- Private Consumption Health (CFH)
- Private Consumption other than Education and Health (CFX)
- Public Consumption Education (CGE)
- Public Consumption Health (CGH)
- Public Consumption other than Education and Health (CGX)
- Public Transfers Pensions, Inflows (TGSOAI)
- Public Transfers Other Cash Inflows (TGXCI)
- Public Transfers Other In-Kind Inflows (TGXII)
- Public Transfers Education Inflows (TGEI)
- Public Transfers Health Inflows (TGHI)
- Public Transfers Outflows (TGO)
- Net Interhousehold Transfers (TFB)
- Net Intrahousehold Transfers (TFW)
- Private Saving (SF)
- Public Saving (SG)
- Labor Income (LY)
- Private Asset Income (YAF)
- Public Asset Income (YAG)

The module also contains parameters for published NTA data (by age only, and by age and sex) allowing for comparing simulation results. Users can choose which NTA data to use in the simulation.

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Dimensions
////////////////////////////////////////////////////////////////////////////////////////////
//

classification NTA_POP_GROUP        //EN NTA Population Group
{
    NPG_CHILD,                      //EN Child 0-16
    NPG_YOUNG_STUDENT,              //EN Student 17-25
    NPG_YOUNG_WORK,                 //EN Non-student 17-25
    NPG_ADULT_WORK,                 //EN Adult 26-59
    NPG_OLD                         //EN Adult 60+
};
```

```
classification NTA_FAMILY_ACTIVE    //EN Family type of non-students 17+ and adults 26-59
{
    NFA_SINGLE_NOKI,                //EN Single childless
    NFA_SINGLE_YOKI,                //EN Single young child(ren)
    NFA_SINGLE_OLKI,                //EN Single older child(ren)
    NFA_COUPLE_NOKI,                //EN Couple childless
    NFA_COUPLE_YOKI,                //EN Couple young child(ren)
    NFA_COUPLE_OLKI                 //EN Couple older child(ren)
};

classification NTA_FAMILY_OLD       //EN Family type of adults 60+
{
    NFR_SINGLE_NOKI,                //EN Single childless
    NFR_SINGLE_KI,                  //EN Single ever parent
    NFR_COUPLE_NOKI,                //EN Couple childless
    NFR_COUPLE_KI                   //EN Couple ever parent
};

range NTA_AGE_CHILD { 0, 16 };      //EN NTA child age range
range NTA_AGE_YOUNG { 17, 25 };     //EN NTA young age range
range NTA_AGE_WORK { 26, 59 };      //EN NTA prime work age range
range NTA_AGE_OLD { 60, 105 };      //EN NTA old age range

classification NTA_VARIABLES        //EN NTA Variables
{
    CFE,            //EN Private Consumption Education (CFE)
    CFH,            //EN Private Consumption Health (CFH)
    CFX,            //EN Private Consumption other than Education and Health (CFX)
    CGE,            //EN Public Consumption Education (CGE)
    CGH,            //EN Public Consumption Health (CGH)
    CGX,            //EN Public Consumption other than Education and Health (CGX)
    TGSOAI,         //EN Public Transfers Pensions, Inflows (TGSOAI)
    TGXCI,          //EN Public Transfers Other Cash Inflows (TGXCI)
    TGXII,          //EN Public Transfers Other In-Kind Inflows (TGXII)
    TGEI,           //EN Public Transfers Education Inflows (TGEI)
    TGHI,           //EN Public Transfers Health Inflows (TGHI)
    TGO,            //EN Public Transfers Outflows (TGO)
    TFB,            //EN Net Interhousehold Transfers (TFB)
    TFW,            //EN Net Intrahousehold Transfers (TFW)
    SF,             //EN Private Saving (SF)
    SG,             //EN Public Saving (SG)
    YL,             //EN Labor Income (LY)
    YAF,            //EN Private Asset Income (YAF)
    YAG             //EN Public Asset Income (YAG)
};

classification NTA_OWN_EDUC //EN Own education
{
    NOE_LOW,        //EN Low
    NOE_MEDIUM,     //EN Medium
    NOE_HIGH        //EN High
};

classification NTA_PAR_EDUC  //EN Parents education
{
    NPE_LOW,        //EN Low
    NPE_MEDIUM,     //EN Medium
    NPE_HIGH        //EN High
};

classification NTA_SCENARIO //EN NTA scenario
{
    NS_AGE,         //EN NTA by age
    NS_SEX,         //EN NTA by age and sex
    NS_FAM          //EN NTA by age, sex, education and family type
};
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN NTA Scenario Selection
    NTA_SCENARIO NtaScenario;

    //EN NTA Children 0-16
    double NtaChild[NTA_PAR_EDUC][NTA_AGE_CHILD][NTA_VARIABLES];

    //EN NTA Students 17-25
    double NtaYoungStudent[NTA_PAR_EDUC][NTA_AGE_YOUNG][NTA_VARIABLES];

    //EN NTA Non-Students 17-25
    double NtaYoungWork[SEX][NTA_OWN_EDUC][NTA_FAMILY_ACTIVE][NTA_AGE_YOUNG][NTA_VARIABLES];

    //EN NTA Adults 26-59
    double NtaAdultWorkAge[SEX][NTA_OWN_EDUC][NTA_FAMILY_ACTIVE][NTA_AGE_WORK][NTA_VARIABLES];

    //EN NTA Adults 60+
    double NtaAdultOldAge[SEX][NTA_OWN_EDUC][NTA_FAMILY_OLD][NTA_AGE_OLD][NTA_VARIABLES];

    //EN NTA by age only
    double NtaAge[AGE_RANGE][NTA_VARIABLES];

    //EN NTA by age and sex only
    double NtaSex[SEX][AGE_RANGE][NTA_VARIABLES];
};

parameter_group PG_NTA_Published                            //EN Published NTA data
{
    NtaAge, NtaSex
};

parameter_group PG_NTA_Refined                              //EN NTA by family and education
{

    NtaChild, NtaYoungStudent, NtaYoungWork,
    NtaAdultWorkAge, NtaAdultOldAge
};

parameter_group PG_NTA                                      //EN NTA Variables
{
    NtaScenario,
    PG_NTA_Published,
    PG_NTA_Refined
};

////////////////////////////////////////////////////////////////////////////////////////////////
/
// Actor
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    //EN NTA Population Group
    NTA_POP_GROUP nta_pop_group =
        (WITHIN(NTA_AGE_CHILD,integer_age)) ? NPG_CHILD :
        (WITHIN(NTA_AGE_YOUNG,integer_age) && is_student) ? NPG_YOUNG_STUDENT :
        (WITHIN(NTA_AGE_YOUNG,integer_age)) ? NPG_YOUNG_WORK :
```

```
        (WITHIN(NTA_AGE_WORK,integer_age)) ? NPG_ADULT_WORK : NPG_OLD;

    //EN Student
    logical is_student = (educ_status == ES_FULLTIME || educ_status == ES_PARTTIME
        || educ_status == ES_DUAL || in_other_education) ? TRUE : FALSE;

    //EN Age Index
    integer nta_age_index = (WITHIN(NTA_AGE_CHILD, integer_age)) ? integer_age :
        (WITHIN(NTA_AGE_YOUNG, integer_age)) ? integer_age - MIN(NTA_AGE_YOUNG) :
        (WITHIN(NTA_AGE_WORK, integer_age)) ? integer_age - MIN(NTA_AGE_WORK)
        : integer_age - MIN(NTA_AGE_OLD);

    //EN NTA Education
    EDUC_LEVEL3 nta_educ = (nta_pop_group == NPG_CHILD || nta_pop_group == NPG_YOUNG_STUDENT) ?
        educ_parents : educ_fate;

    //EN NTA Family Type Index Workers
    integer nta_fam_index = (nta_pop_group == NPG_YOUNG_WORK || nta_pop_group == NPG_ADULT_WORK) ?
        (
        (!has_partner) ?
            (
            (sex == FEMALE && age_youngest_child < 3) ? NFA_SINGLE_YOKI :
                (sex == FEMALE && age_youngest_child < 18) ? NFA_SINGLE_OLKI :
                NFA_SINGLE_NOKI
                )
            :
            (
            ((sex == FEMALE && age_youngest_child < 3) || (sex == MALE && lSpouse-
>age_youngest_child < 3)) ? NFA_COUPLE_YOKI :
                ((sex == FEMALE && age_youngest_child < 18) || (sex == MALE && lSpouse-
>age_youngest_child < 18)) ? NFA_COUPLE_OLKI :
                NFA_COUPLE_NOKI
                )
            ) : 0;


     integer nta_old_index = (nta_pop_group == NPG_OLD) ?
        (
             (!has_partner) ?
                (
                    ((sex==MALE && never_father) || (sex==FEMALE && !is_mother)) ?
NFR_SINGLE_NOKI : NFR_SINGLE_KI
                )
            :
                (
                    ((sex==MALE && never_father) || (sex==FEMALE && !is_mother)) ?
NFR_COUPLE_NOKI : NFR_COUPLE_KI
                )
        )
        : 0;


        //EN Private Consumption Education (CFE)
        double base_CFE =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][CFE] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][CFE] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][CFE] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][CFE] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][CFE] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][CFE] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][CFE];
        //EN Private Consumption Health (CFH)
```

```
        double base_CFH =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][CFH] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][CFH] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][CFH] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][CFH] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][CFH] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][CFH] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][CFH];

        //EN Private Consumption other than Education and Health (CFX)
        double base_CFX =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][CFX] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][CFX] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][CFX] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][CFX] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][CFX] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][CFX] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][CFX];

        //EN Public Consumption Education (CGE)
        double base_CGE =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][CGE] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][CGE] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][CGE] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][CGE] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][CGE] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][CGE] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][CGE];

        //EN Public Consumption Health (CGH)
        double base_CGH =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][CGH] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][CGH] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][CGH] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][CGH] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][CGH] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][CGH] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][CGH];

        //EN Public Consumption other than Education and Health (CGX)
        double base_CGX =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][CGX] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][CGX] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][CGX] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][CGX] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][CGX] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][CGX] :
```

```
NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][CGX];

        //EN Public Transfers Pensions, Inflows (TGSOAI)
        double base_TGSOAI =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][TGSOAI] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][TGSOAI] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][TGSOAI] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][TGSOAI] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][TGSOAI] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][TGSOAI] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][TGSOAI];

        //EN Public Transfers Other Cash Inflows (TGXCI)
        double base_TGXCI =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][TGXCI] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][TGXCI] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][TGXCI] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][TGXCI] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][TGXCI] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][TGXCI] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][TGXCI];

        //EN Public Transfers Other In-Kind Inflows (TGXII)
        double base_TGXII =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][TGXII] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][TGXII] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][TGXII] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][TGXII] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][TGXII] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][TGXII] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][TGXII];

        //EN Public Transfers Health Outflows (TGEI)
        double base_TGEI =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][TGEI] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][TGEI] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][TGEI] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][TGEI] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][TGEI] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][TGEI] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][TGEI];

        //EN Public Transfers Health Outflows (TGHI)
        double base_TGHI =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][TGHI] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][TGHI] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][TGHI] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][TGHI] :
```

```
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][TGHI] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][TGHI] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][TGHI];


        //EN Public Transfers Education Outflows (TGO)
        double base_TGO =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][TGO] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][TGO] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][TGO] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][TGO] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][TGO] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][TGO] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][TGO];

        //EN Net Interhousehold Transfers (TFB)
        double base_TFB =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][TFB] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][TFB] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][TFB] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][TFB] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][TFB] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][TFB] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][TFB];

        //EN Net Intrahousehold Transfers (TFW)
        double base_TFW =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][TFW] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][TFW] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][TFW] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][TFW] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][TFW] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][TFW] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][TFW];

        //EN Private Saving (SF)
        double base_SF =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][SF] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][SF] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][SF] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?  NtaYoungStudent[nta_educ][nta_age_index][SF] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][SF] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][SF] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][SF];

        //EN Public Saving (SG)
        double base_SG =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][SG] :
```

```
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][SG] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][SG] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?  NtaYoungStudent[nta_educ][nta_age_index][SG] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][SG] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][SG] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][SG];

        //EN Labor Inclome (LY)
        double base_YL =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][YL] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][YL] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][YL] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?  NtaYoungStudent[nta_educ][nta_age_index][YL] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][YL] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][YL] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][YL];

        //EN Private Asset Income (YAF)
        double base_YAF =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][YAF] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][YAF] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][YAF] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][YAF] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][YAF] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][YAF] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][YAF];

        //EN Public Asset Income (YAG)
        double base_YAG =
            (NtaScenario == NS_AGE) ?               NtaAge[integer_age][YAG] :
            (NtaScenario == NS_SEX) ?               NtaSex[sex][integer_age][YAG] :
            (nta_pop_group == NPG_CHILD) ?          NtaChild[nta_educ][nta_age_index][YAG] :
            (nta_pop_group == NPG_YOUNG_STUDENT) ?
NtaYoungStudent[nta_educ][nta_age_index][YAG] :
            (nta_pop_group == NPG_YOUNG_WORK) ?
NtaYoungWork[sex][nta_educ][nta_fam_index][nta_age_index][YAG] :
            (nta_pop_group == NPG_ADULT_WORK) ?
NtaAdultWorkAge[sex][nta_educ][nta_fam_index][nta_age_index][YAG] :

NtaAdultOldAge[sex][nta_educ][nta_old_index][nta_age_index][YAG];
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_BasicNtaTabs //EN Basic NTA Tables
{
    TabBasicNta, TabBasicNtaByAge
};

actor Person
{
    SIM_YEAR_RANGE sim_year = COERCE(SIM_YEAR_RANGE, calendar_year); //EN Year
```

```
};

table Person TabBasicNta                        //EN Basic NTA totals
[WITHIN(SIM_YEAR_RANGE, calendar_year) && is_resident]
{
    {
        duration(),                             //EN Population
        weighted_duration(base_CFE),            //EN Private Consumption Education (CFE)
        weighted_duration(base_CFH),            //EN Private Consumption Health (CFH)
        weighted_duration(base_CFX),            //EN Private Consumption other than Education and
Health (CFX)
        weighted_duration(base_CGE),            //EN Public Consumption Education (CGE)
        weighted_duration(base_CGH),            //EN Public Consumption Health (CGH)
        weighted_duration(base_CGX),            //EN Public Consumption other than Education and
Health (CGX)
        weighted_duration(base_TGSOAI),         //EN Public Transfers Pensions, Inflows (TGSOAI)
        weighted_duration(base_TGXCI),          //EN Public Transfers Other Cash Inflows (TGXCI)
        weighted_duration(base_TGXII),          //EN Public Transfers Other In-Kind Inflows
(TGXII)
        weighted_duration(base_TGEI),           //EN Public Transfers Education Inflows (TGEI)
        weighted_duration(base_TGHI),           //EN Public Transfers Health Inflows (TGHI)
        weighted_duration(base_TGO),            //EN Public Transfers Outflows (TGO)
        weighted_duration(base_TFB),            //EN Net Interhousehold Transfers (TFB)
        weighted_duration(base_TFW),            //EN Net Intrahousehold Transfers (TFW)
        weighted_duration(base_SF),             //EN Private Saving (SF)
        weighted_duration(base_SG),             //EN Public Saving (SG)
        weighted_duration(base_YL),             //EN Labor Income (LY)
        weighted_duration(base_YAF),            //EN Private Asset Income (YAF)
        weighted_duration(base_YAG)             //EN Public Asset Income (YAG)
    }
    * sim_year
};

table Person TabBasicNtaByAge                   //EN NTA by age
[WITHIN(SIM_YEAR_RANGE, calendar_year) && is_resident]
{
    sim_year*
    {
        duration(),                                         //EN Population
        weighted_duration(base_CFE) / duration(),           //EN Private Consumption Education
(CFE)
        weighted_duration(base_CFH) / duration(),           //EN Private Consumption Health (CFH)
        weighted_duration(base_CFX) / duration(),           //EN Private Consumption other than
Education and Health (CFX)
        weighted_duration(base_CGE) / duration(),           //EN Public Consumption Education
(CGE)
        weighted_duration(base_CGH) / duration(),           //EN Public Consumption Health (CGH)
        weighted_duration(base_CGX) / duration(),           //EN Public Consumption other than
Education and Health (CGX)
        weighted_duration(base_TGSOAI) / duration(),        //EN Public Transfers Pensions,
Inflows (TGSOAI)
        weighted_duration(base_TGXCI) / duration(),         //EN Public Transfers Other Cash
Inflows (TGXCI)
        weighted_duration(base_TGXII) / duration(),         //EN Public Transfers Other In-Kind
Inflows (TGXII)
        weighted_duration(base_TGEI) / duration(),          //EN Public Transfers Education
Inflows (TGEI)
        weighted_duration(base_TGHI) / duration(),          //EN Public Transfers Health Inflows
(TGHI)
        weighted_duration(base_TGO) / duration(),           //EN Public Transfers Outflows (TGO)
        weighted_duration(base_TFB) / duration(),           //EN Net Interhousehold Transfers
(TFB)
        weighted_duration(base_TFW) / duration(),           //EN Net Intrahousehold Transfers
(TFW)
        weighted_duration(base_SF) / duration(),            //EN Private Saving (SF)
        weighted_duration(base_SG) / duration(),            //EN Public Saving (SG)
```

```
        weighted_duration(base_YL) / duration(),            //EN Labor Income (LY)
        weighted_duration(base_YAF) / duration(),           //EN Private Asset Income (YAF)
        weighted_duration(base_YAG) / duration()            //EN Public Asset Income (YAG)
    }
    * integer_age
};

table Person tabLCDperCap                                   //EN Life Course Deficit
[WITHIN(SIM_YEAR_RANGE, calendar_year) && is_resident]
{
    {
        (weighted_duration(base_CFE) + weighted_duration(base_CFH) + weighted_duration(base_CFX) +
weighted_duration(base_CGE) + weighted_duration(base_CGH) + weighted_duration(base_CGX)) /
duration(), //EN average consumption
        weighted_duration(base_YL) / duration(),            //EN Average labor Income (LY)
        -(weighted_duration(base_CFE) + weighted_duration(base_CFH) + weighted_duration(base_CFX)
+ weighted_duration(base_CGE) + weighted_duration(base_CGH) + weighted_duration(base_CGX) -
weighted_duration(base_YL)) / duration() //EN Average gap
    }
* sim_year
};
```

Step 18: Basic NTA Indicators

Overview

At this step we add a module for the calculation of basic NTA indicators - the Support Ratio and the Impact Index as suggested by Lee & Mason (2017). Indices are calculated and updated in yearly steps on the State level. To do so, we add an actor “State” performing these updates, and a module StateCore.mpp.

The new NtaIndicators.mpp Module

This module calculates a set of yearly NTA indicators following the approach presented by Lee & Mason (2017) “Some Economic Impacts of Changing Population Age Distributions—Capital, Labor and Transfers”. We model a simple economy with a Cobb Douglas production function without productivity growth in two scenarios: a closed economy where interest rate and wages are endogenous, and a open economy where wages and interest rates stay constant. The two indicators calculated are the “Support Ratio” and the “Impact Index”. Both measures have the “Effective consumers” - i.e. the age re-weighted consumption based on the reference year - in the denominator. The nominator of the Support Ratio is the projected total labor. In contrast, the “Impact Index” puts the total consumption assuming constant saving rates by age and accounting for the changes in wages and interest rates into the nominator.:

```
Variables:

 - yl(x)   Average labor income age x
 - yk(x)   Average capital income age x
 - P(x)    Population age x
 - i(x)    Average savings age x
 - s(x)    Saving rate age x - constant
 - c(x)    Average Consumption age x (reference values for calculation of N) - reference

 - L       Labor
 - K       Capital
 - I       Saving
 - S       Saving rate
 - C       Consumption
 - r       Interest rate
 - w       Wage
 - Yl      Labor Income
 - YK      Capital Income
 - Y       Total Income Yl+Yk
 - α       Alpha – constant
 - N       Effective Consumers (population-weighted base-year consumption)
 - l(x)    Average labor age x – constant
 - k(x)    Average capital age x – constant

 Cobb Douglas:

 Y = L^α K^(1- α)
```

```
Y = wL + rK
w = αY / L
r = (1- α) Y / K
Yl = wL = αY
Yk = rK = (1- α)Y

Known:

yl(x)   NTA data of reference year
yk(x)   NTA data of reference year
i(x)    NTA data of reference year
c(x)    NTA data of reference year
P(x)    NTA data of reference year
r       parameter for reference year (used also to estimate stock from capital income flow)

Calculated for initial year:

Yl   = ∑ yl(x) * P(x)
Yk   = ∑ yk(x) * P(x)
Y    = Yl + Yk
α    = Yl / Y
K    = Yk / r
L    = ( Y / K^(1- α))^(1/α)
w    = Yl / L
s(x) = i(x) / (yl(x) + ykx))
l(x) = yl (x) / w
k(x) = yk (x) / r
N    = ∑ cl(x) * P(x)

Simulation: calculate for an updated population by age P(x)

Closed Economy

L     = ∑ l(x)* P(x)
K     = ∑ k(x)* P(x)
Y     = L^α K^(1- α)
Yl    = αY
YK    = (1- α)Y
w     = αY / L
r     = (1- α) Y / K
N     = ∑ c(x) * P(x)
yl(x) = w * l(x)
yk(x) = r * k(x)
C     = ∑ (1-s(x)) * (yl(x) + yk(x) ) * P(x)

Open Economy (difference to closed)

Y     = w * L + r * K
Yl    = w * L
YK    = r * K

Indices

SR    = L / N  Support Ratio (same for open and closed economy)
IMP   = C / N  Impact Index (different for closed and open economy)
```

This module is optional. It feeds on the NTA variables (by age, sex, education, family) calculated and maintained in the NtaBase.mpp module. The only model parameters are the reference year of the NTA data and the initial interest rate. All calculations are performed in a yearly update event of the actor State.

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////////
//

parameter_group PG_NTA_CONST                        //EN NTA Indicators
{
    NtaInitialInterestRate, NtaBaseYear
};

parameters
{
    int    NtaBaseYear;                            //EN NTA Base Year
    double NtaInitialInterestRate;                 //EN Initial Interest Rate
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// State Actor states and functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor State
{
    double  nta_initial_Y = { 0.0 };               //EN Initial year total income Y
    double  nta_initial_Yl = { 0.0 };              //EN Initial year labor income Yl
    double  nta_initial_Yk = { 0.0 };              //EN Initial year capital income Yk
    double  nta_initial_K = { 0.0 };               //EN Initial year capital stock
    double  nta_alpha = { 0.0 };                   //EN Alpha of Cobb Douglas Function
    double  nta_initial_L = { 0.0 };               //EN Initial year total labor L
    double  nta_initial_w = { 0.0 };               //EN Initial year wage w
    double  nta_initial_SR = { 1.0 };              //EN Initial Support Ratio
    double  nta_initial_closed_IMP = { 1.0 };      //EN Initial Impact Factor closed economy
    double  nta_initial_open_IMP = { 1.0 };        //EN Initial Impact Factor open economy

    double  nta_current_N = { 0.0 };               //EN Current reference consumers N
    double  nta_current_L = { 0.0 };               //EN Current total Labor N
    double  nta_current_C = { 0.0 };               //EN Current total Consumption
    double  nta_current_C_open = { 0.0 };          //EN Current total Consumption (open economy)
    double  nta_current_Y = { 0.0 };               //EN Current total Y
    double  nta_current_K = { 0.0 };               //EN Current total Capital K
    double  nta_current_w = { 0.0 };               //EN Current wage w
    double  nta_current_r = { 0.0 };               //EN Current interest rate r
    double  nta_current_SR = { 1.0 };              //EN Current Support Ratio
    double  nta_current_closed_IMP = { 1.0 };      //EN Current Impact Factor closed economy
    double  nta_current_open_IMP = { 1.0 };        //EN Current Impact Factor open economy

    logical nta_is_updated = { FALSE };

    TIME    time_nta_update = { 0.0 };             //EN Time of next NTA update
    event   timeUpdateNtaEvent, UpdateNtaEvent;    //EN Update NTA production function

    SIM_YEAR_RANGE tab_state_year = COERCE(SIM_YEAR_RANGE, state_year); //EN Calendar Year
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

TIME State::timeUpdateNtaEvent()
{
    if (time_nta_update < NtaBaseYear ) return NtaBaseYear + 0.5;
```

```
    else return time_nta_update;
}

void State::UpdateNtaEvent()
{
    nta_is_updated = FALSE;

    // Population size
    //long nPersons = asAllPersons->Count();
    long nPersons = asAllResidents->Count(); //CHANGE19
//    nta_total_pop = nPersons;

    if (state_year == NtaBaseYear)
    {
        // Values of initial year
        for (long nI = 0; nI < nPersons; nI++)
        {
            //auto prPerson = asAllPersons->Item(nI);
            auto prPerson = asAllResidents->Item(nI);//CHANGE19
            nta_initial_Yl = nta_initial_Yl + prPerson->base_YL;
            nta_initial_Yk = nta_initial_Yk + prPerson->base_YAF + prPerson->base_YAG;
        }
        nta_initial_Y = nta_initial_Yl + nta_initial_Yk;
        nta_initial_K = nta_initial_Yk / NtaInitialInterestRate;
        nta_alpha = nta_initial_Yl / nta_initial_Y;
        nta_initial_L = pow(nta_initial_Y / pow(nta_initial_K,1-nta_alpha), 1.0 / nta_alpha);
        nta_initial_w = nta_initial_Yl / nta_initial_L;
    }

    if (state_year >= NtaBaseYear)
    {
        double dNta_sx = 0.0;          // applicable individual saving rate as in initial year
        double dNta_lx = 0.0;          // applicable individual labor input as in initial year
        double dNta_kx = 0.0;          // applicable individual capital as in initial year
        double dNta_cx = 0.0;          // applicable individual consumption as in initial year

        nta_current_L = 0.0;           // Current total Labor N
        nta_current_K = 0.0;           // Current total Capital K
        nta_current_N = 0.0;           // Current reference consumers N
        nta_current_C = 0.0;           // Current total Consumption
        nta_current_C_open = 0.0;      // Current total Consumption in open economy
        nta_current_Y = 0.0;           // Current total Y

        for (long nI = 0; nI < nPersons; nI++)
        {
            //auto prPerson = asAllPersons->Item(nI);
            auto prPerson = asAllResidents->Item(nI);//CHANGE19
            dNta_lx = prPerson->base_YL / nta_initial_w;
            dNta_kx = (prPerson->base_YAF + prPerson->base_YAG) / NtaInitialInterestRate;
            dNta_cx = prPerson->base_CFE + prPerson->base_CFH + prPerson->base_CFX + prPerson-
>base_CGE
                + prPerson->base_CGH + prPerson->base_CGX;
            nta_current_L = nta_current_L + dNta_lx;
            nta_current_K = nta_current_K + dNta_kx;
            nta_current_N = nta_current_N + dNta_cx;
        }
        nta_current_Y = pow(nta_current_L, nta_alpha) * pow(nta_current_K, 1 - nta_alpha);
        nta_current_w = nta_alpha * nta_current_Y / nta_current_L;
        nta_current_r = (1 - nta_alpha) * nta_current_Y / nta_current_K;

        for (long nI = 0; nI < nPersons; nI++)
        {
            //auto prPerson = asAllPersons->Item(nI);
            auto prPerson = asAllResidents->Item(nI);//CHANGE19
            dNta_lx = prPerson->base_YL / nta_initial_w;
            dNta_kx = (prPerson->base_YAF + prPerson->base_YAG) / NtaInitialInterestRate;
```

```
            dNta_sx = 0.0;
            if (prPerson->base_YL + prPerson->base_YAF + prPerson->base_YAG > 0.0 && prPerson-
>base_SF + prPerson->base_SG > 0.0 &&
                prPerson->base_YL + prPerson->base_YAF + prPerson->base_YAG > prPerson->base_SF +
prPerson->base_SG)
            {
                dNta_sx = (prPerson->base_SF + prPerson->base_SG) / (prPerson->base_YL + prPerson-
>base_YAF + prPerson->base_YAG);
            }
            nta_current_C = nta_current_C + (1 - dNta_sx) * (nta_current_w * dNta_lx +
nta_current_r * dNta_kx);
            nta_current_C_open = nta_current_C_open + (1 - dNta_sx) * (nta_initial_w * dNta_lx +
NtaInitialInterestRate * dNta_kx);
        }
    }

    // Indices

    if (state_year == NtaBaseYear)
    {
        nta_initial_SR = nta_initial_L / nta_current_N;
        nta_initial_closed_IMP = nta_current_C / nta_current_N;
        nta_initial_open_IMP = nta_current_C_open / nta_current_N;

    }

    if (state_year >= NtaBaseYear)
    {
        nta_current_SR = (  nta_current_L / nta_current_N   ) / nta_initial_SR;
        nta_current_closed_IMP = ( nta_current_C / nta_current_N ) / nta_initial_closed_IMP;
        nta_current_open_IMP = ( nta_current_C_open / nta_current_N ) / nta_initial_open_IMP;
    }

    // Update Clock
    nta_is_updated = TRUE;
    time_nta_update = WAIT(1.0);
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Output
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_NTAIndicators                    //EN NTA Indicators and Impact Index Lee 2017
{
    TabNTAIndicators
};

table State TabNTAIndicators                    //EN NTA Indicators and Impact Index Lee 2017
[WITHIN(SIM_YEAR_RANGE,state_year) && nta_is_updated]
{
    {
        max_value_in(nta_current_SR),           //EN Support Ratio SR decimals=4
        max_value_in(nta_current_closed_IMP),   //EN Impact Index IMP closed economy decimals=4
        max_value_in(nta_current_open_IMP),     //EN Impact Index IMP open economy decimals=4
        max_value_in(nta_current_N),            //EN Current reference consumers N
        max_value_in(nta_current_L),            //EN Current total Labor L
        max_value_in(nta_current_C),            //EN Current total Consumption
        max_value_in(nta_current_Y),            //EN Current total Income Y
        max_value_in(nta_current_K),            //EN Current total Capital K
        max_value_in(nta_current_w),            //EN Current wage w closed economy decimals=4
        max_value_in(nta_current_r)             //EN Current interest rate r closed economy
decimals=4
    }
    * tab_state_year
```

```
};
```

The new StateCore.mpp Module

The welfare state core actor file introduces a ‘State’ actor. The actor maintains a state_year which is synchronized with the clock actor. The “State” is used for welfare state accounting

```
actor State                                          //EN State Actor
{
    void    Start();                                 //EN Start Function
    int     state_year = lClock->clock_year;         //EN Calendar year
    void    StateYearStart();                        //EN Year End Function
};

link State.lClock Clock.lState;                      //EN Link State to Clock
actor_set State asState;                             //EN The State Actor Set

void State::Start()
{
    // Setting the actor weight
    Set_actor_weight(asGlobals->Item(0)->person_weight);
    Set_actor_subsample_weight(asGlobals->Item(0)->person_weight);

    lClock = asTheClock->Item(0);
    time = lClock->time;
}
```

Initiating the State actor in model_core.mpp

The actor State has to be created after the clock actor (to which it links itself in order to synchronizes time)

```
    // Create the Clock
    auto prClock = new Clock();
    prClock->Start();

    // added at step 16:
    // Create the State
    auto prState = new State();
    prState->Start();
```

Step 19: NTA Validation

Overview

At this step we add an optional module for the validation of NTA variables.

The new NtaValidation.mpp Module

The NTA Validation Module is an optional module introducing a set of validation tables comparing the aggregate outcome of NTA variables by education and family type with published aggregates by sex a/o age. For each of the 19 NTA variables, a table is produced comparing aggregated simulated values in the starting year with published NTA values and providing aggregate simulated outcomes by education group. Also a table of the population composition by all considered dimensions (age, sex, education, family type) is produced. This module can be removed to save memory space.

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Dymensions
////////////////////////////////////////////////////////////////////////////////////////////
//

range TAB_FAM_INDEX { 0,5 }; //EN Family Index

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states
////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    //EN Family Index
    TAB_FAM_INDEX tab_fam_index = (nta_pop_group == NPG_OLD) ?
        COERCE(TAB_FAM_INDEX, nta_old_index) : COERCE(TAB_FAM_INDEX, nta_fam_index);

    // NTA by age only
    double age_CFE = NtaAge[integer_age][CFE];              //EN CFE by age
    double age_CFH = NtaAge[integer_age][CFH];              //EN CFH by age
    double age_CFX = NtaAge[integer_age][CFX];              //EN CFX by age
    double age_CGE = NtaAge[integer_age][CGE];              //EN CGE by age
    double age_CGH = NtaAge[integer_age][CGH];              //EN CGH by age
    double age_CGX = NtaAge[integer_age][CGX];              //EN CGX by age
    double age_TGSOAI = NtaAge[integer_age][TGSOAI];        //EN TGSOAI by age
    double age_TGXCI = NtaAge[integer_age][TGXCI];          //EN TGXCI by age
    double age_TGXII = NtaAge[integer_age][TGXII];          //EN TGXII by age
    double age_TGEI = NtaAge[integer_age][TGEI];            //EN TGEI by age
    double age_TGHI = NtaAge[integer_age][TGHI];            //EN TGHI by age
    double age_TGO = NtaAge[integer_age][TGO];              //EN TGO by age
    double age_TFB = NtaAge[integer_age][TFB];              //EN TFB by age
    double age_TFW = NtaAge[integer_age][TFW];              //EN TFW by age
    double age_SF = NtaAge[integer_age][SF];                //EN SF by age
    double age_SG = NtaAge[integer_age][SG];                //EN SG by age
```

```
    double age_YL = NtaAge[integer_age][YL];                //EN LY by age
    double age_YAF = NtaAge[integer_age][YAF];              //EN YAF by age
    double age_YAG = NtaAge[integer_age][YAG];              //EN YAG by age

    // NTA by age and sex
    double sex_CFE = NtaSex[sex][integer_age][CFE];         //EN CFE by age and sex
    double sex_CFH = NtaSex[sex][integer_age][CFH];         //EN CFH by age and sex
    double sex_CFX = NtaSex[sex][integer_age][CFX];         //EN CFX by age and sex
    double sex_CGE = NtaSex[sex][integer_age][CGE];         //EN CGE by age and sex
    double sex_CGH = NtaSex[sex][integer_age][CGH];         //EN CGH by age and sex
    double sex_CGX = NtaSex[sex][integer_age][CGX];         //EN CGX by age and sex
    double sex_TGSOAI = NtaSex[sex][integer_age][TGSOAI];   //EN TGSOAI by age and sex
    double sex_TGXCI = NtaSex[sex][integer_age][TGXCI];     //EN TGXCI by age and sex
    double sex_TGXII = NtaSex[sex][integer_age][TGXII];     //EN TGXII by age and sex
    double sex_TGEI = NtaSex[sex][integer_age][TGEI];       //EN TGEI by age and sex
    double sex_TGHI = NtaSex[sex][integer_age][TGHI];       //EN TGHI by age and sex
    double sex_TGO = NtaSex[sex][integer_age][TGO];         //EN TGO by age and sex
    double sex_TFB = NtaSex[sex][integer_age][TFB];         //EN TFB by age and sex
    double sex_TFW = NtaSex[sex][integer_age][TFW];         //EN TFW by age and sex
    double sex_SF = NtaSex[sex][integer_age][SF];           //EN SF by age and sex
    double sex_SG = NtaSex[sex][integer_age][SG];           //EN SG by age and sex
    double sex_YL = NtaSex[sex][integer_age][YL];           //EN LY by age and sex
    double sex_YAF = NtaSex[sex][integer_age][YAF];         //EN YAF by age and sex
    double sex_YAG = NtaSex[sex][integer_age][YAG];         //EN YAG by age and sex
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Validation Tables
////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_NTA_ValidationTables     //EN NTA Validation Tables
{
    TabNtaValidation_CFE, TabNtaValidation_CFH, TabNtaValidation_CFX, TabNtaValidation_CGE,
    TabNtaValidation_CGH, TabNtaValidation_CGX, TabNtaValidation_TGSOAI, TabNtaValidation_TGXCI,
    TabNtaValidation_TGXII, TabNtaValidation_TGEI, TabNtaValidation_TGHI, TabNtaValidation_TGO,
    TabNtaValidation_TFB, TabNtaValidation_TFW, TabNtaValidation_SF, TabNtaValidation_SG,
    TabNtaValidation_YL, TabNtaValidation_YAF, TabNtaValidation_YAG, TabNtaPopulation2010
};

table Person TabNtaValidation_CFE                                       //EN CFE
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CFE) / duration(),                       //EN CFE
        weighted_duration(sex,FEMALE, base_CFE) / duration(sex,FEMALE),         //EN CFE female
        weighted_duration(sex,MALE, base_CFE) / duration(sex,MALE),             //EN CFE male

        weighted_duration(nta_educ,EL3_LOW, base_CFE) / duration(nta_educ,EL3_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_CFE) / duration(nta_educ,EL3_MEDIUM),  //EN
CFE educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_CFE) / duration(nta_educ,EL3_HIGH),    //EN CFE
educ high

        weighted_duration(sex,FEMALE,sex_CFE) / duration(sex,FEMALE),           //EN Agenta CFE
female
        weighted_duration(sex,MALE,sex_CFE) / duration(sex,MALE),               //EN Agenta CFE
male
        weighted_duration(age_CFE) / duration()                         //EN Agenta CFE by
age
    }
    * integer_age
};
```

```
table Person TabNtaValidation_CFH                                          //EN CFH
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CFH) / duration(),                          //EN CFH
        weighted_duration(sex,FEMALE, base_CFH) / duration(sex,FEMALE),    //EN CFH female
        weighted_duration(sex,MALE, base_CFH) / duration(sex,MALE),        //EN CFH male

        weighted_duration(nta_educ,EL3_LOW, base_CFH) / duration(nta_educ,EL3_LOW),     //EN CFH
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_CFH) / duration(nta_educ,EL3_MEDIUM),  //EN
CFH educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_CFH) / duration(nta_educ,EL3_HIGH),    //EN CFH
educ high

        weighted_duration(sex,FEMALE,sex_CFH) / duration(sex,FEMALE),      //EN Agenta CFH
female
        weighted_duration(sex,MALE,sex_CFH) / duration(sex,MALE),          //EN Agenta CFH
male
        weighted_duration(age_CFH) / duration()                            //EN Agenta CFH by
age
    }
    * integer_age
};

table Person TabNtaValidation_CFX                                          //EN CFX
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CFX) / duration(),                          //EN CFX
        weighted_duration(sex,FEMALE, base_CFX) / duration(sex,FEMALE),    //EN CFX female
        weighted_duration(sex,MALE, base_CFX) / duration(sex,MALE),        //EN CFX male

        weighted_duration(nta_educ,EL3_LOW, base_CFX) / duration(nta_educ,EL3_LOW),     //EN CFX
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_CFX) / duration(nta_educ,EL3_MEDIUM),  //EN
CFX educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_CFX) / duration(nta_educ,EL3_HIGH),    //EN CFX
educ high

        weighted_duration(sex,FEMALE,sex_CFX) / duration(sex,FEMALE),      //EN Agenta CFX
female
        weighted_duration(sex,MALE,sex_CFX) / duration(sex,MALE),          //EN Agenta CFX
male
        weighted_duration(age_CFX) / duration()                            //EN Agenta CFX by
age
    }
    * integer_age
};

table Person TabNtaValidation_CGE                                          //EN CGE
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CGE) / duration(),                          //EN CGE
        weighted_duration(sex,FEMALE, base_CGE) / duration(sex,FEMALE),    //EN CGE female
        weighted_duration(sex,MALE, base_CGE) / duration(sex,MALE),        //EN CGE male

        weighted_duration(nta_educ,EL3_LOW, base_CGE) / duration(nta_educ,EL3_LOW),     //EN CGE
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_CGE) / duration(nta_educ,EL3_MEDIUM),  //EN
CGE educ medium
```

```
        weighted_duration(nta_educ,EL3_HIGH, base_CGE) / duration(nta_educ,EL3_HIGH),    //EN CGE
educ high

        weighted_duration(sex,FEMALE,sex_CGE) / duration(sex,FEMALE),           //EN Agenta CGE
female
        weighted_duration(sex,MALE,sex_CGE) / duration(sex,MALE),               //EN Agenta CGE
male
        weighted_duration(age_CGE) / duration()                                 //EN Agenta CGE by
age
    }
    * integer_age
};


table Person TabNtaValidation_CGH                                               //EN CGH
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CGH) / duration(),                               //EN CGH
        weighted_duration(sex,FEMALE, base_CGH) / duration(sex,FEMALE),         //EN CGH female
        weighted_duration(sex,MALE, base_CGH) / duration(sex,MALE),             //EN CGH male

        weighted_duration(nta_educ,EL3_LOW, base_CGH) / duration(nta_educ,EL3_LOW),     //EN CGH
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_CGH) / duration(nta_educ,EL3_MEDIUM),  //EN
CGH educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_CGH) / duration(nta_educ,EL3_HIGH),    //EN CGH
educ high

        weighted_duration(sex,FEMALE,sex_CGH) / duration(sex,FEMALE),           //EN Agenta CGH
female
        weighted_duration(sex,MALE,sex_CGH) / duration(sex,MALE),               //EN Agenta CGH
male
        weighted_duration(age_CGH) / duration()                                 //EN Agenta CGH by
age
    }
    * integer_age
};


table Person TabNtaValidation_CGX                                               //EN CGX
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CGX) / duration(),                               //EN CGX
        weighted_duration(sex,FEMALE, base_CGX) / duration(sex,FEMALE),         //EN CGX female
        weighted_duration(sex,MALE, base_CGX) / duration(sex,MALE),             //EN CGX male

        weighted_duration(nta_educ,EL3_LOW, base_CGX) / duration(nta_educ,EL3_LOW),     //EN CGX
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_CGX) / duration(nta_educ,EL3_MEDIUM),  //EN
CGX educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_CGX) / duration(nta_educ,EL3_HIGH),    //EN CGX
educ high

        weighted_duration(sex,FEMALE,sex_CGX) / duration(sex,FEMALE),           //EN Agenta CGX
female
        weighted_duration(sex,MALE,sex_CGX) / duration(sex,MALE),               //EN Agenta CGX
male
        weighted_duration(age_CGX) / duration()                                 //EN Agenta CGX by
age
    }
    * integer_age
};
```

```
table Person TabNtaValidation_TGSOAI                                          //EN TGSOAI
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGSOAI) / duration(),                          //EN TGSOAI
        weighted_duration(sex,FEMALE, base_TGSOAI) / duration(sex,FEMALE),    //EN TGSOAI
female
        weighted_duration(sex,MALE, base_TGSOAI) / duration(sex,MALE),        //EN TGSOAI
male

        weighted_duration(nta_educ,EL3_LOW, base_TGSOAI) / duration(nta_educ,EL3_LOW),     //EN
TGSOAI educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_TGSOAI) / duration(nta_educ,EL3_MEDIUM),  //EN
TGSOAI educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_TGSOAI) / duration(nta_educ,EL3_HIGH),    //EN
TGSOAI educ high

        weighted_duration(sex,FEMALE,sex_TGSOAI) / duration(sex,FEMALE),      //EN Agenta
TGSOAI female
        weighted_duration(sex,MALE,sex_TGSOAI) / duration(sex,MALE),          //EN Agenta
TGSOAI male
        weighted_duration(age_TGSOAI) / duration()                            //EN Agenta
TGSOAI by age
    }
    * integer_age
};

table Person TabNtaValidation_TGXCI                                           //EN TGXCI
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGXCI) / duration(),                           //EN TGXCI
        weighted_duration(sex,FEMALE, base_TGXCI) / duration(sex,FEMALE),     //EN TGXCI
female
        weighted_duration(sex,MALE, base_TGXCI) / duration(sex,MALE),         //EN TGXCI male

        weighted_duration(nta_educ,EL3_LOW, base_TGXCI) / duration(nta_educ,EL3_LOW),     //EN
TGXCI educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_TGXCI) / duration(nta_educ,EL3_MEDIUM),  //EN
TGXCI educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_TGXCI) / duration(nta_educ,EL3_HIGH),    //EN
TGXCI educ high

        weighted_duration(sex,FEMALE,sex_TGXCI) / duration(sex,FEMALE),       //EN Agenta
TGXCI female
        weighted_duration(sex,MALE,sex_TGXCI) / duration(sex,MALE),           //EN Agenta
TGXCI male
        weighted_duration(age_TGXCI) / duration()                             //EN Agenta
TGXCI by age
    }
    * integer_age
};

table Person TabNtaValidation_TGXII                                           //EN TGXII
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGXII) / duration(),                           //EN TGXII
        weighted_duration(sex,FEMALE, base_TGXII) / duration(sex,FEMALE),     //EN TGXII
female
        weighted_duration(sex,MALE, base_TGXII) / duration(sex,MALE),         //EN TGXII male
```

```
        weighted_duration(nta_educ,EL3_LOW, base_TGXII) / duration(nta_educ,EL3_LOW),     //EN
TGXII educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_TGXII) / duration(nta_educ,EL3_MEDIUM),  //EN
TGXII educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_TGXII) / duration(nta_educ,EL3_HIGH),    //EN
TGXII educ high

        weighted_duration(sex,FEMALE,sex_TGXII) / duration(sex,FEMALE),           //EN Agenta
TGXII female
        weighted_duration(sex,MALE,sex_TGXII) / duration(sex,MALE),               //EN Agenta
TGXII male
        weighted_duration(age_TGXII) / duration()                                 //EN Agenta
TGXII by age
    }
    * integer_age
};

table Person TabNtaValidation_TGEI                                                //EN TGEI
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGEI) / duration(),                                //EN TGEI
        weighted_duration(sex,FEMALE, base_TGEI) / duration(sex,FEMALE),          //EN TGEI female
        weighted_duration(sex,MALE, base_TGEI) / duration(sex,MALE),              //EN TGEI male

        weighted_duration(nta_educ,EL3_LOW, base_TGEI) / duration(nta_educ,EL3_LOW),     //EN TGEI
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_TGEI) / duration(nta_educ,EL3_MEDIUM),  //EN
TGEI educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_TGEI) / duration(nta_educ,EL3_HIGH),    //EN
TGEI educ high

        weighted_duration(sex,FEMALE,sex_TGEI) / duration(sex,FEMALE),            //EN Agenta TGEI
female
        weighted_duration(sex,MALE,sex_TGEI) / duration(sex,MALE),                //EN Agenta TGEI
male
        weighted_duration(age_TGEI) / duration()                                  //EN Agenta TGEI
by age
    }
    * integer_age
};

table Person TabNtaValidation_TGHI                                                //EN TGHI
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGHI) / duration(),                                //EN TGHI
        weighted_duration(sex,FEMALE, base_TGHI) / duration(sex,FEMALE),          //EN TGHI female
        weighted_duration(sex,MALE, base_TGHI) / duration(sex,MALE),              //EN TGHI male

        weighted_duration(nta_educ,EL3_LOW, base_TGHI) / duration(nta_educ,EL3_LOW),     //EN TGHI
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_TGHI) / duration(nta_educ,EL3_MEDIUM),  //EN
TGHI educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_TGHI) / duration(nta_educ,EL3_HIGH),    //EN
TGHI educ high

        weighted_duration(sex,FEMALE,sex_TGHI) / duration(sex,FEMALE),            //EN Agenta TGHI
female
        weighted_duration(sex,MALE,sex_TGHI) / duration(sex,MALE),                //EN Agenta TGHI
male
        weighted_duration(age_TGHI) / duration()                                  //EN Agenta TGHI
by age
    }
```

```
    * integer_age
};

table Person TabNtaValidation_TGO                                                    //EN TGO
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGO) / duration(),                                    //EN TGO
        weighted_duration(sex,FEMALE, base_TGO) / duration(sex,FEMALE),              //EN TGO female
        weighted_duration(sex,MALE, base_TGO) / duration(sex,MALE),                  //EN TGO male

        weighted_duration(nta_educ,EL3_LOW, base_TGO) / duration(nta_educ,EL3_LOW),     //EN TGO
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_TGO) / duration(nta_educ,EL3_MEDIUM),  //EN
TGO educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_TGO) / duration(nta_educ,EL3_HIGH),    //EN TGO
educ high

        weighted_duration(sex,FEMALE,sex_TGO) / duration(sex,FEMALE),                //EN Agenta TGO
female
        weighted_duration(sex,MALE,sex_TGO) / duration(sex,MALE),                    //EN Agenta TGO
male
        weighted_duration(age_TGO) / duration()                                      //EN Agenta TGO by
age
    }
    * integer_age
};

table Person TabNtaValidation_TFB                                                    //EN TFB
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TFB) / duration(),                                    //EN TFB
        weighted_duration(sex,FEMALE, base_TFB) / duration(sex,FEMALE),              //EN TFB female
        weighted_duration(sex,MALE, base_TFB) / duration(sex,MALE),                  //EN TFB male

        weighted_duration(nta_educ,EL3_LOW, base_TFB) / duration(nta_educ,EL3_LOW),     //EN TFB
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_TFB) / duration(nta_educ,EL3_MEDIUM),  //EN
TFB educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_TFB) / duration(nta_educ,EL3_HIGH),    //EN TFB
educ high

        weighted_duration(sex,FEMALE,sex_TFB) / duration(sex,FEMALE),                //EN Agenta TFB
female
        weighted_duration(sex,MALE,sex_TFB) / duration(sex,MALE),                    //EN Agenta TFB
male
        weighted_duration(age_TFB) / duration()                                      //EN Agenta TFB by
age
    }
    * integer_age
};

table Person TabNtaValidation_TFW                                                    //EN TFW
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TFW) / duration(),                                    //EN TFW
        weighted_duration(sex,FEMALE, base_TFW) / duration(sex,FEMALE),              //EN TFW female
        weighted_duration(sex,MALE, base_TFW) / duration(sex,MALE),                  //EN TFW male

        weighted_duration(nta_educ,EL3_LOW, base_TFW) / duration(nta_educ,EL3_LOW),     //EN TFW
educ low
```

```
        weighted_duration(nta_educ,EL3_MEDIUM, base_TFW) / duration(nta_educ,EL3_MEDIUM),  //EN
TFW educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_TFW) / duration(nta_educ,EL3_HIGH),    //EN TFW
educ high

        weighted_duration(sex,FEMALE,sex_TFW) / duration(sex,FEMALE),           //EN Agenta TFW
female
        weighted_duration(sex,MALE,sex_TFW) / duration(sex,MALE),               //EN Agenta TFW
male
        weighted_duration(age_TFW) / duration()                                 //EN Agenta TFW by
age
    }
    * integer_age
};

table Person TabNtaValidation_SF                                                //EN SF Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_SF) / duration(),                                //EN SF
        weighted_duration(sex,FEMALE, base_SF) / duration(sex,FEMALE),          //EN SF female
        weighted_duration(sex,MALE, base_SF) / duration(sex,MALE),              //EN SF male

        weighted_duration(nta_educ,EL3_LOW, base_SF) / duration(nta_educ,EL3_LOW),     //EN SF
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_SF) / duration(nta_educ,EL3_MEDIUM),  //EN SF
educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_SF) / duration(nta_educ,EL3_HIGH),    //EN SF
educ high

        weighted_duration(sex,FEMALE,sex_SF) / duration(sex,FEMALE),            //EN Agenta SF
female
        weighted_duration(sex,MALE,sex_SF) / duration(sex,MALE),                //EN Agenta SF male
        weighted_duration(age_SF) / duration()                                  //EN Agenta SF by
age
    }
    * integer_age
};

table Person TabNtaValidation_SG                                                //EN SG Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_SG) / duration(),                                //EN SG
        weighted_duration(sex,FEMALE, base_SG) / duration(sex,FEMALE),          //EN SG female
        weighted_duration(sex,MALE, base_SG) / duration(sex,MALE),              //EN SG male

        weighted_duration(nta_educ,EL3_LOW, base_SG) / duration(nta_educ,EL3_LOW),     //EN SG
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_SG) / duration(nta_educ,EL3_MEDIUM),  //EN SG
educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_SG) / duration(nta_educ,EL3_HIGH),    //EN SG
educ high

        weighted_duration(sex,FEMALE,sex_SG) / duration(sex,FEMALE),            //EN Agenta SG
female
        weighted_duration(sex,MALE,sex_SG) / duration(sex,MALE),                //EN Agenta SG male
        weighted_duration(age_SG) / duration()                                  //EN Agenta SG by
age
    }
    * integer_age
};

table Person TabNtaValidation_YL                                                //EN YL Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
```

```
    {
        weighted_duration(base_YL) / duration(),                                    //EN YL
        weighted_duration(sex,FEMALE, base_YL) / duration(sex,FEMALE),              //EN YL female
        weighted_duration(sex,MALE, base_YL) / duration(sex,MALE),                  //EN YL male

        weighted_duration(nta_educ,EL3_LOW, base_YL) / duration(nta_educ,EL3_LOW),     //EN YL
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_YL) / duration(nta_educ,EL3_MEDIUM),  //EN YL
educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_YL) / duration(nta_educ,EL3_HIGH),    //EN YL
educ high

        weighted_duration(sex,FEMALE,sex_YL) / duration(sex,FEMALE),           //EN Agenta YL
female
        weighted_duration(sex,MALE,sex_YL) / duration(sex,MALE),               //EN Agenta YL male
        weighted_duration(age_YL) / duration()                                 //EN Agenta YL by
age
    }
    * integer_age
};

table Person TabNtaValidation_YAF                                               //EN YAF
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_YAF) / duration(),                               //EN YAF
        weighted_duration(sex,FEMALE, base_YAF) / duration(sex,FEMALE),         //EN YAF female
        weighted_duration(sex,MALE, base_YAF) / duration(sex,MALE),             //EN YAF male

        weighted_duration(nta_educ,EL3_LOW, base_YAF) / duration(nta_educ,EL3_LOW),     //EN YAF
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_YAF) / duration(nta_educ,EL3_MEDIUM),  //EN
YAF educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_YAF) / duration(nta_educ,EL3_HIGH),    //EN YAF
educ high

        weighted_duration(sex,FEMALE,sex_YAF) / duration(sex,FEMALE),           //EN Agenta YAF
female
        weighted_duration(sex,MALE,sex_YAF) / duration(sex,MALE),               //EN Agenta YAF
male
        weighted_duration(age_YAF) / duration()                                 //EN Agenta YAF by
age
    }
    * integer_age
};

table Person TabNtaValidation_YAG                                               //EN YAG
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_YAG) / duration(),                               //EN YAG
        weighted_duration(sex,FEMALE, base_YAG) / duration(sex,FEMALE),         //EN YAG female
        weighted_duration(sex,MALE, base_YAG) / duration(sex,MALE),             //EN YAG male

        weighted_duration(nta_educ,EL3_LOW, base_YAG) / duration(nta_educ,EL3_LOW),     //EN YAG
educ low
        weighted_duration(nta_educ,EL3_MEDIUM, base_YAG) / duration(nta_educ,EL3_MEDIUM),  //EN
YAG educ medium
        weighted_duration(nta_educ,EL3_HIGH, base_YAG) / duration(nta_educ,EL3_HIGH),    //EN YAG
educ high

        weighted_duration(sex,FEMALE,sex_YAG) / duration(sex,FEMALE),           //EN Agenta YAG
female
```

```
        weighted_duration(sex,MALE,sex_YAG) / duration(sex,MALE),               //EN Agenta YAG
male
        weighted_duration(age_YAG) / duration()                                 //EN Agenta YAG by
age
    }
    * integer_age
};


table Person TabNtaPopulation2010                           //EN NTA POPULATION
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    nta_pop_group *
    tab_fam_index + *
    sex + *
    {
        duration() //EN Duration
    }
    * integer_age
    * nta_educ +
};
```

Step 20: NTA Full Generational Accounts

Overview

At this step we add the optional module NtaFullGenAccounts.mpp for the calculation of Full Generational Accounts (FGAs) as suggested by Lee et.al. (2017).

The new NtaFullGenAccounts.mpp Module

This module introduces Full Generational Accounts (FGAs) as suggested by Lee et.al. 2017 in: Full Generational Accounts: What Do We Give to the Next Generation?

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    double  FgaEconomicGrowth;                                  //EN Economic growth
    double  FgaDiscountFactor;                                  //EN Discount factor
    model_generated double mgFgaUprateFactor[SIM_YEAR_RANGE];   //EN Uprate factor

    //EN Discount factor for present value at birth
    model_generated double mgFgaDeflationFactor[AGE_RANGE];

    //EN Discount factor for present value 2010
    model_generated double mgFgaDeflation2010[SIM_YEAR_RANGE];

    //EN Discount factor for present value 2040
    model_generated double mgFgaDeflation2040[SIM_YEAR_RANGE];
};

parameter_group PG_FGA                                          //EN Full Generational Accounts
{
    FgaEconomicGrowth, FgaDiscountFactor
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Person
////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    // Age in tables
    AGE_RANGE age_2010 = COERCE(AGE_RANGE,2010 - year_of_birth);        //EN Age
    AGE_RANGE age_2011 = COERCE(AGE_RANGE,2011 - year_of_birth);        //EN Age
    AGE_RANGE age_2040 = COERCE(AGE_RANGE,2040 - year_of_birth);        //EN Age


    real fga_YL = base_YL * mgFgaUprateFactor[RANGE_POS(SIM_YEAR_RANGE,calendar_year)];
```

```
    real fga_TF = (base_TFB + base_TFW) *
mgFgaUprateFactor[RANGE_POS(SIM_YEAR_RANGE,calendar_year)];
    real fga_TGI = (base_TGSOAI + base_TGXCI + base_TGXII + base_TGEI +  base_TGHI)
        * mgFgaUprateFactor[RANGE_POS(SIM_YEAR_RANGE,calendar_year)];
    real fga_TGO = base_TGO * mgFgaUprateFactor[RANGE_POS(SIM_YEAR_RANGE,calendar_year)];
    real fga_TF_adj = (fga_TF > 0.0) ? fga_TF * lGlobals->adj_factor_TFI
        : fga_TF * lGlobals->adj_factor_TFO;
    real fga_TGI_adj = fga_TGI * lGlobals->adj_factor_TGI;
    real fga_TGO_adj = fga_TGO * lGlobals->adj_factor_TGO;

    real deflation_factor = mgFgaDeflationFactor[integer_age];

    // PV Age 0

    real fga_YL_defl = deflation_factor * fga_YL;
    real fga_YL_pv = weighted_duration(fga_YL_defl);

    real fga_TF_defl = deflation_factor * fga_TF;
    real fga_TF_pv = weighted_duration(fga_TF_defl);

    real fga_TGI_defl = deflation_factor * fga_TGI;
    real fga_TGI_pv = weighted_duration(fga_TGI_defl);

    real fga_TGO_defl = deflation_factor * fga_TGO;
    real fga_TGO_pv = weighted_duration(fga_TGO_defl);

    real fga_TF_adj_defl = deflation_factor * fga_TF_adj;
    real fga_TF_adj_pv = weighted_duration(fga_TF_adj_defl);

    real fga_TGI_adj_defl = deflation_factor * fga_TGI_adj;
    real fga_TGI_adj_pv = weighted_duration(fga_TGI_adj_defl);

    real fga_TGO_adj_defl = deflation_factor * fga_TGO_adj;
    real fga_TGO_adj_pv = weighted_duration(fga_TGO_adj_defl);

    // PV 2010 (only transfers 2010+)

    real deflation_2010 = (calendar_year >= MIN(SIM_YEAR_RANGE)) ?
        mgFgaDeflation2010[RANGE_POS(SIM_YEAR_RANGE,calendar_year)] : 0.0;

    real fga_YL_defl_2010 = deflation_2010 * fga_YL;
    real fga_YL_pv_2010 = weighted_duration(fga_YL_defl_2010);

    real fga_TF_defl_2010 = deflation_2010 * fga_TF;
    real fga_TF_pv_2010 = weighted_duration(fga_TF_defl_2010);

    real fga_TGI_defl_2010 = deflation_2010 * fga_TGI;
    real fga_TGI_pv_2010 = weighted_duration(fga_TGI_defl_2010);

    real fga_TGO_defl_2010 = deflation_2010 * fga_TGO;
    real fga_TGO_pv_2010 = weighted_duration(fga_TGO_defl_2010);

    real fga_TF_adj_defl_2010 = deflation_2010 * fga_TF_adj;
    real fga_TF_adj_pv_2010 = weighted_duration(fga_TF_adj_defl_2010);

    real fga_TGI_adj_defl_2010 = deflation_2010 * fga_TGI_adj;
    real fga_TGI_adj_pv_2010 = weighted_duration(fga_TGI_adj_defl_2010);

    real fga_TGO_adj_defl_2010 = deflation_2010 * fga_TGO_adj;
    real fga_TGO_adj_pv_2010 = weighted_duration(fga_TGO_adj_defl_2010);

     // PV 2040 (only transfers 2040+)

    real deflation_2040 = (calendar_year >= 2040) ?
        mgFgaDeflation2040[RANGE_POS(SIM_YEAR_RANGE,calendar_year)] : 0.0;
```

```
    real fga_YL_defl_2040 = deflation_2040 * fga_YL;
    real fga_YL_pv_2040 = weighted_duration(fga_YL_defl_2040);

    real fga_TF_defl_2040 = deflation_2040 * fga_TF;
    real fga_TF_pv_2040 = weighted_duration(fga_TF_defl_2040);

    real fga_TGI_defl_2040 = deflation_2040 * fga_TGI;
    real fga_TGI_pv_2040 = weighted_duration(fga_TGI_defl_2040);

    real fga_TGO_defl_2040 = deflation_2040 * fga_TGO;
    real fga_TGO_pv_2040 = weighted_duration(fga_TGO_defl_2040);

    real fga_TF_adj_defl_2040 = deflation_2040 * fga_TF_adj;
    real fga_TF_adj_pv_2040 = weighted_duration(fga_TF_adj_defl_2040);

    real fga_TGI_adj_defl_2040 = deflation_2040 * fga_TGI_adj;
    real fga_TGI_adj_pv_2040 = weighted_duration(fga_TGI_adj_defl_2040);

    real fga_TGO_adj_defl_2040 = deflation_2040 * fga_TGO_adj;
    real fga_TGO_adj_pv_2040 = weighted_duration(fga_TGO_adj_defl_2040);
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor Globals
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Globals
{
    real adj_factor_TFI = {1.0};
    real adj_factor_TFO = {1.0};
    real adj_factor_TGI = {1.0};
    real adj_factor_TGO = {1.0};
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor State
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor State
{
    TIME time_update_fga = { 0.0 };                         //EN Time of next FGA udjustment
update
    event timeUpdateFgaAdjustEvent, UpdateFgaAdjustEvent;   //EN FGA adjustment update event
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

TIME State::timeUpdateFgaAdjustEvent()
{
    if (time_update_fga < MIN(SIM_YEAR_RANGE)) return MIN(SIM_YEAR_RANGE) + 0.5;
    else return time_update_fga;
}

void State::UpdateFgaAdjustEvent()
{
    // Population size
    long nPersons = asAllResidents->Count();
```

```
    double dTotalTFI = 0.0;
    double dTotalTFO = 0.0;
    double dTotalTGI = 0.0;
    double dTotalTGO = 0.0;
    double dAverage;

    // Loop adding individual NTAs
    for (long nI = 0; nI < nPersons; nI++)
    {
        auto prPerson = asAllResidents->Item(nI);
        if (prPerson->fga_TF > 0.0) dTotalTFI = dTotalTFI + prPerson->fga_TF;
        else dTotalTFO = dTotalTFO - prPerson->fga_TF;
        dTotalTGI = dTotalTGI + prPerson->fga_TGI;
        dTotalTGO = dTotalTGO + prPerson->fga_TGO;
    }
    // Adjust TG
    dAverage = (dTotalTGI + dTotalTGO) / 2.0;
    lStateToGlobals->adj_factor_TGI = dAverage / dTotalTGI;
    lStateToGlobals->adj_factor_TGO = dAverage / dTotalTGO;

    // Adjust TF
    dAverage = (dTotalTFI + dTotalTFO) / 2.0;
    lStateToGlobals->adj_factor_TFI = dAverage / dTotalTFI;
    lStateToGlobals->adj_factor_TFO = dAverage / dTotalTFO;

    // Program next update event
    time_update_fga = WAIT(1.0);
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Pre-Simulation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void PreSimulation()
{
    for (int nJ = 0; nJ < SIZE(SIM_YEAR_RANGE); nJ++)
    {
        mgFgaUprateFactor[nJ] = 1.0 * pow((1.0 + FgaEconomicGrowth), nJ);
    }

    for (int nJ = 0; nJ < SIZE(AGE_RANGE); nJ++)
    {
        mgFgaDeflationFactor[nJ] = 1.0 * pow((1.0 / (1.0 + FgaDiscountFactor)), nJ);
    }

    for (int nJ = 0; nJ < SIZE(SIM_YEAR_RANGE); nJ++)
    {
        if (MIN(SIM_YEAR_RANGE) + nJ >= 2010) mgFgaDeflation2010[nJ] = 1.0
            * pow((1.0 / (1.0 + FgaDiscountFactor)), 2010-MIN(SIM_YEAR_RANGE)+nJ);
        else mgFgaDeflation2010[nJ] = 0.0;
    }

    for (int nJ = 0; nJ < SIZE(SIM_YEAR_RANGE); nJ++)
    {
        if (MIN(SIM_YEAR_RANGE) + nJ >= 2040) mgFgaDeflation2040[nJ] = 1.0
            * pow((1.0 / (1.0 + FgaDiscountFactor)), 2040-MIN(SIM_YEAR_RANGE)+nJ);
        else mgFgaDeflation2040[nJ] = 0.0;
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
```

```
////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_FullGenerationalAccounts             //EN NTA Full Generational Accounts
{
    tabFgaUnadjustedTotals, tabFgaPV,
    tabAdjustmentFga,
    tabFgaPV_2010, tabFgaPV_2040
};

table Person tabFgaUnadjustedTotals                 //EN FGA Totals
[is_resident && in_projected_time]
{
    {
        weighted_duration(fga_YL),                  //EN YL total
        weighted_duration(fga_TF),                  //EN TF total
        weighted_duration(fga_TGI),                 //EN TGI total
        weighted_duration(fga_TGO),                 //EN TGO total
        weighted_duration(fga_TF_adj),              //EN TF total adjusted
        weighted_duration(fga_TGI_adj),             //EN TGI total adjusted
        weighted_duration(fga_TGO_adj)              //EN TGO total adjusted
    }
    * sim_year
};

table Person tabFgaPV                               //EN FGA PV
[is_resident && person_type == PT_CHILD
&& trigger_transitions(is_alive,TRUE,FALSE)]
{
    educ_fate+ *
    {
        value_in(fga_YL_pv) / unit,                 //EN Present Value YL at birth
        value_in(fga_TF_pv) / unit,                 //EN Present Value TF at birth
        value_in(fga_TGI_pv) / unit,                //EN Present Value TGI at birth
        value_in(fga_TGO_pv) / unit,                //EN Present Value TGO at birth
        value_in(fga_TF_adj_pv) / unit,             //EN Present Value TF adjusted at birth
        value_in(fga_TGI_adj_pv) / unit,            //EN Present Value TGI adjsuted at birth
        value_in(fga_TGO_adj_pv) / unit             //EN Present Value TGO adjusted at birth
    }
    * year_of_birth
};


table Globals tabAdjustmentFga                      //EN FGA Adjustment factors
[globals_in_projected_time]
{
    {
        max_value_in(adj_factor_TFI),               //EN Adjustment TFI decimals=3
        max_value_in(adj_factor_TFO),               //EN Adjustment TFO decimals=3
        max_value_in(adj_factor_TGI),               //EN Adjustment TGI decimals=3
        max_value_in(adj_factor_TGO)                //EN Adjustment TGO decimals=3
    }
    * global_tab_sim_year
};



table Person tabFgaPV_2010                          //EN FGA PV 2010
[is_resident && person_type != PT_IMMIGRANT
&& trigger_transitions(is_alive,TRUE,FALSE)
&& year_of_birth <= 2010]
{
    educ_fate+ *
    {
        value_in(fga_YL_pv_2010) / unit,            //EN Present Value YL 2010
        value_in(fga_TF_pv_2010) / unit,            //EN Present Value TF 2010
```

```
        value_in(fga_TGI_pv_2010) / unit,           //EN Present Value TGI 2010
        value_in(fga_TGO_pv_2010) / unit,           //EN Present Value TGO 2010
        value_in(fga_TF_adj_pv_2010) / unit,        //EN Present Value TF adjusted 2010
        value_in(fga_TGI_adj_pv_2010) / unit,       //EN Present Value TGI adjusted 2010
        value_in(fga_TGO_adj_pv_2010) / unit        //EN Present Value TGO adjusted 2010
    }
    * age_2010
};


table Person tabFgaPV_2011                          //EN FGA PV 2011
[is_resident && person_type != PT_IMMIGRANT
&& trigger_transitions(is_alive,TRUE,FALSE)
&& year_of_birth <= 2011]
{
    educ_fate+ *
    {
        value_in(fga_YL_pv_2010) / unit,            //EN Present Value YL 2010
        value_in(fga_TF_pv_2010) / unit,            //EN Present Value TF 2010
        value_in(fga_TGI_pv_2010) / unit,           //EN Present Value TGI 2010
        value_in(fga_TGO_pv_2010) / unit,           //EN Present Value TGO 2010
        value_in(fga_TF_adj_pv_2010) / unit,        //EN Present Value TF adjusted 2010
        value_in(fga_TGI_adj_pv_2010) / unit,       //EN Present Value TGI adjusted 2010
        value_in(fga_TGO_adj_pv_2010) / unit        //EN Present Value TGO adjusted 2010
    }
    * age_2011
};

table Person tabFgaPV_2040                          //EN FGA PV 2040
[is_resident && person_type != PT_IMMIGRANT
&& trigger_transitions(is_alive,TRUE,FALSE)
&& year_of_birth <= 2040]
{
    educ_fate+ *
    {
        value_in(fga_YL_pv_2040) / unit,            //EN Present Value YL 2040
        value_in(fga_TF_pv_2040) / unit,            //EN Present Value TF 2040
        value_in(fga_TGI_pv_2040) / unit,           //EN Present Value TGI 2040
        value_in(fga_TGO_pv_2040) / unit,           //EN Present Value TGO 2040
        value_in(fga_TF_adj_pv_2040) / unit,        //EN Present Value TF adjusted 2040
        value_in(fga_TGI_adj_pv_2040) / unit,       //EN Present Value TGI adjusted 2040
        value_in(fga_TGO_adj_pv_2040) / unit        //EN Present Value TGO adjusted 2040
    }
    * age_2040
};
```

Step 21: Net Migration

Overview

At this step we add the optional module NetMigration.mpp which gives the user an additional option of modeling international migration - net migration - based on a parameter of total net migration by ag, sex, and period. This option is useful if only data on net migration are available, as is the case for Eurostat population projections.

The new NetMigration.mpp Module

This module gives the user an additional option of modeling international migration - net migration - based on a parameter of total net migration by age, sex, and period. This option is useful if only data on net migration are available, as is the case for Eurostat population projections. The module introduces a new selection switch for modeling international migration:

- Do not model migration
- Immigration only
- Emigration only
- Immigration & Emigration
- Net Migration

This selection parameter is used to set the previousely used checkboxes for modeling immigration and emigration which were changed to model-generated parameters set in the pre-simulation phase. If net migration is chosen, the immigration parameters are overwritten by values based on the new net migration parameter. For emigration, a new Clock function DoNetEmigration() was added which - when selected - replaces the DoEmigration() function.

The addition of this module required only minor changes to other modules:

- Emigration.mpp: the parameter ModelEmigration is made model-generated
- Immigration.mpp the parameter ModelImmigration is made model-generated, ImmigrationAgeSexAll is changed from cumrate to double
- model_core.mpp: the condition to generate immigrants is changed to also include net immigration
- PersonCore: age and sex of immigrants are now sampled from the model-generated parameter MgImmigrationAgeSexAll

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Types
////////////////////////////////////////////////////////////////////////////////////////////////
//


classification MIGRATION_SETTINGS //EN Migration Settings
{
    MSE_NON,            //EN Do not model migration
    MSE_IMMI,           //EN Immigration only
    MSE_EMI,            //EN Emigration only
    MSE_ALL,            //EN Immigration & Emigration
    MSE_NET             //EN Net Migration
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Parameters
////////////////////////////////////////////////////////////////////////////////////////////////
//

parameters
{
    //EN Migration Settings
    MIGRATION_SETTINGS MigrationSettings;

    //EN Model net migration
    model_generated logical ModelNetMigration;

    //EN Net migration by age and sex
    double NetMigrationSexPeriodAge[SEX][SIM_YEAR_RANGE][AGE_RANGE];

    //EN Total number of immigrants
    model_generated long MgImmigrationTotal[SIM_YEAR_RANGE];

    //EN Age-Sex distribution of immigrants
    model_generated cumrate [2] MgImmigrationAgeSexAll[SIM_YEAR_RANGE][SEX][AGE_RANGE];
};

parameter_group PG_NetMigration                          //EN Net Migration
{
    NetMigrationSexPeriodAge
};

parameter_group PG_Migration                            //EN International Migration
{
    MigrationSettings,
    PG_Immigration,
    PG_Emigration,
    PG_NetMigration
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// States & Functions
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Clock
{
    void DoNetEmigration();                                 //EN Net Emigration
    hook DoNetEmigration, MidYearClockEvent, 6;             //EN hook to mid year
};
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void Clock::DoNetEmigration()
{
    if (clock_year >= MIN(SIM_YEAR_RANGE) && ModelNetMigration == TRUE)
    {
        // for each age by sex
        for (int nSex = 0; nSex < SIZE(SEX); nSex++)
        {
            for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
            {
                double dExpectedEmigrants = 0.0;
                long nExpectedEmigrants = 0;

                if (NetMigrationSexPeriodAge[nSex][RANGE_POS(SIM_YEAR_RANGE, clock_year)][nAge] <
0) // there is net emigration
                {
                    // determine number of emigrants
                    dExpectedEmigrants = -NetMigrationSexPeriodAge[nSex][RANGE_POS(SIM_YEAR_RANGE,
clock_year)][nAge]
                        / asGlobals->Item(0)->person_weight;
                    nExpectedEmigrants = int(dExpectedEmigrants);
                    if (RandUniform(46) < dExpectedEmigrants - (double)nExpectedEmigrants)
nExpectedEmigrants++;

                    // Make somebody emigrate
                    for (int nIndex = 0; nIndex < nExpectedEmigrants; nIndex++)
                    {
                        if (asAllPersonsSexAge[nSex][nAge]->Count() > 0)
                        {
                            auto prPerson = asAllPersonsSexAge[nSex][nAge]-
>GetRandom(RandUniform(47));
                            prPerson->DoEmigrate();
                        }
                    }
                }
            }
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Pre-Simulation
////////////////////////////////////////////////////////////////////////////////////////////////
//

void PreSimulation()
{
    // Settings

    if (MigrationSettings == MSE_NON)                   // Do not model migration
    {
        ModelNetMigration = FALSE;
        ModelImmigration = FALSE;
        ModelEmigration = FALSE;
    }
    else if (MigrationSettings == MSE_IMMI)             //EN Immigration only
    {
        ModelNetMigration = FALSE;
        ModelImmigration = TRUE;
```

```
        ModelEmigration = FALSE;
    }
    else if (MigrationSettings == MSE_EMI)              //EN Emigration only
    {
        ModelNetMigration = FALSE;
        ModelImmigration = FALSE;
        ModelEmigration = TRUE;
    }
    else if (MigrationSettings == MSE_ALL)              //EN Immigration & Emigration
    {
        ModelNetMigration = FALSE;
        ModelImmigration = TRUE;
        ModelEmigration = TRUE;
    }
    else                                                //EN Net Migration
    {
        ModelNetMigration = TRUE;
        ModelImmigration = FALSE;
        ModelEmigration = FALSE;
    }

    // Parameters

    for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
    {
        MgImmigrationTotal[nYear] = 0.0;
    }

    if (ModelNetMigration)
    {
        for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
        {
            double dSumImmigrants = 0;
            for (int nSex = 0; nSex < SIZE(SEX); nSex++)
            {
                for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
                {
                    if (NetMigrationSexPeriodAge[nSex][nYear][nAge] > 0)
                    {
                        MgImmigrationAgeSexAll[nYear][nSex][nAge] =
NetMigrationSexPeriodAge[nSex][nYear][nAge];
                        dSumImmigrants = dSumImmigrants +
NetMigrationSexPeriodAge[nSex][nYear][nAge];
                    }
                    else
                    {
                        MgImmigrationAgeSexAll[nYear][nSex][nAge] = 0.0;
                    }
                }
            }
            MgImmigrationTotal[nYear] = dSumImmigrants;
        }
    }
    else
    {
        for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
        {
            MgImmigrationTotal[nYear] = ImmigrationTotal[nYear];
        }

        for (int nYear = 0; nYear < SIZE(SIM_YEAR_RANGE); nYear++)
        {
            for (int nSex = 0; nSex < SIZE(SEX); nSex++)
            {
                for (int nAge = 0; nAge < SIZE(AGE_RANGE); nAge++)
                {
```

```
                    MgImmigrationAgeSexAll[nYear][nSex][nAge] = ImmigrationAgeSexAll[nSex][nAge];
                }
            }
        }
    }
}
```

Step 22: Table Output

Overview

This step reorganizes and substantially extendes the model's table output. Also It finalizes the model's self-documentation. The result is the final model version used in the studies of the WELTRANSIM Work Package 2 report. It is also the model version available for download at the project website.

The TablesEducation.mpp Module

This module implements a set of tables related to education.

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Dimensions
////////////////////////////////////////////////////////////////////////////////////////////////
//

partition AGE_530 { 5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30 };
//EN Age

classification YEAR10       //EN Year
{
    YEAR_2010,              //EN 2010
    YEAR_2020,              //EN 2020
    YEAR_2030,              //EN 2030
    YEAR_2040,              //EN 2040
    YEAR_2050,              //EN 2050
    YEAR_2060,              //EN 2060
    YEAR_2070,              //EN 2070
    YEAR_2080,              //EN 2080
    YEAR_2090,              //EN 2090
    YEAR_2100               //EN 2100
};

classification YEAR05       //EN Year
{
    YEAR5_2010,              //EN 2010
    YEAR5_2015,              //EN 2015
    YEAR5_2020,              //EN 2020
    YEAR5_2025,              //EN 2025
    YEAR5_2030,              //EN 2030
    YEAR5_2035,              //EN 2035
    YEAR5_2040,              //EN 2040
    YEAR5_2045,              //EN 2045
    YEAR5_2050,              //EN 2050
    YEAR5_2055,              //EN 2055
    YEAR5_2060,              //EN 2060
    YEAR5_2065,              //EN 2065
    YEAR5_2070,              //EN 2070
    YEAR5_2075,              //EN 2075
    YEAR5_2080,              //EN 2080
    YEAR5_2085,              //EN 2085
```

```
    YEAR5_2090,              //EN 2090
    YEAR5_2095,              //EN 2095
    YEAR5_2100               //EN 2100
};

partition AGE_5100  //EN Age Groups
{
    5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,
    65, 70, 75, 80, 85, 90, 95, 100
};


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table States
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    //EN Year
    YEAR10 year10 = (calendar_year == 2010) ? YEAR_2010 :
    (calendar_year == 2020) ? YEAR_2020 :
    (calendar_year == 2030) ? YEAR_2030 :
    (calendar_year == 2040) ? YEAR_2040 :
    (calendar_year == 2050) ? YEAR_2050 :
    (calendar_year == 2060) ? YEAR_2060 :
    (calendar_year == 2070) ? YEAR_2070 :
    (calendar_year == 2080) ? YEAR_2080 :
    (calendar_year == 2090) ? YEAR_2090 : YEAR_2100;

    //EN Year
    YEAR05 year05 = (calendar_year == 2010) ? YEAR5_2010 :
    (calendar_year == 2015) ? YEAR5_2015 :
    (calendar_year == 2020) ? YEAR5_2020 :
    (calendar_year == 2025) ? YEAR5_2025 :
    (calendar_year == 2030) ? YEAR5_2030 :
    (calendar_year == 2035) ? YEAR5_2035 :
    (calendar_year == 2040) ? YEAR5_2040 :
    (calendar_year == 2045) ? YEAR5_2045 :
    (calendar_year == 2050) ? YEAR5_2050 :
    (calendar_year == 2055) ? YEAR5_2055 :
    (calendar_year == 2060) ? YEAR5_2060 :
    (calendar_year == 2065) ? YEAR5_2065 :
    (calendar_year == 2070) ? YEAR5_2070 :
    (calendar_year == 2075) ? YEAR5_2075 :
    (calendar_year == 2080) ? YEAR5_2080 :
    (calendar_year == 2085) ? YEAR5_2085 :
    (calendar_year == 2090) ? YEAR5_2090 :
    (calendar_year == 2095) ? YEAR5_2095 : YEAR5_2100;

    logical is_year05 = (calendar_year == 2010 || calendar_year == 2020 || calendar_year == 2030
|| calendar_year == 2040
        || calendar_year == 2050 || calendar_year == 2060 || calendar_year == 2070
        || calendar_year == 2080 || calendar_year == 2090 || calendar_year == 2100
        || calendar_year == 2015 || calendar_year == 2025 || calendar_year == 2035 ||
calendar_year == 2045
        || calendar_year == 2055 || calendar_year == 2065 || calendar_year == 2075
        || calendar_year == 2085 || calendar_year == 2095) ? TRUE : FALSE;


    logical is_year10 = (calendar_year == 2010 || calendar_year == 2020 || calendar_year == 2030
|| calendar_year == 2040
        || calendar_year == 2050 || calendar_year == 2060 || calendar_year == 2070
        || calendar_year == 2080 || calendar_year == 2090 || calendar_year == 2100) ? TRUE :
FALSE;
```

```
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Groups
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_EDUCATION_TABLES                         //EN Education
{
    tabEducSchoolEnrolment,
    tabPopulationByEducation,
    tabPopulationByEducation2059,
    tabEducationYob,
    tabEducationParentsYob
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table Person tabEducSchoolEnrolment                     //EN School enrolment
[is_resident && in_projected_time]
{
    {
        duration(in_school, TRUE) / duration(),         //EN In any school decimals=2
        duration(in_regular_school, TRUE) / duration(), //EN In regular school decimals=2
        duration(in_other_education, TRUE) / duration() //EN In other school decimals=2
    }
    * split(integer_age, AGE_530)                       //EN Age
    * sim_year
};

table Person tabPopulationByEducation                   //EN Population by age group and education
[is_resident && is_year10]
{
    year10 *
    sex+ *
    {
        duration()
    }
    * split(integer_age, AGE_5100)+                     //EN Age
    * educ3_level+
};

table Person tabPopulationByEducation2059              //EN Population 20-59 by education
[is_resident && is_year05 && integer_age > 19 && integer_age < 60]
{
    sex+ *
    {
        duration()
    }
    * year05
    * educ3_level+
};


table Person tabEducationYob                //EN Education distribution at 30 by year of birth
[is_resident && integer_age == 30]
{
    sex+ *
    {
        duration()
```

```
    }
    * year_of_birth
    * educ_fate+
};

table Person tabEducationParentsYob          //EN Education of parents by year of birth
[is_resident && integer_age == 0]
{
    sex+ *
    {
        duration()
    }
    * year_of_birth
    * educ_parents+
};
```

The TablesFamily.mpp Module

This module implements a set of tables related to families

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Dimensions
////////////////////////////////////////////////////////////////////////////////////////////////
//

partition   SHORT_TIME{ 0.01 };
partition   AGE_1025{ 10, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26 }; //EN Age

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table States
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    //EN Single, no dependent children at home
    logical is_single_alone = (!has_partner && children_in_household == 0) ? TRUE : FALSE;

    //EN Single, dependent children at home
    logical is_single_child = (!has_partner && children_in_household > 0) ? TRUE : FALSE;

    //EN In partnership, no dependent children at home
    logical is_couple_alone = (has_partner && children_in_household == 0) ? TRUE : FALSE;

    //EN In partnership, dependent children at home
    logical is_couple_child = (has_partner && children_in_household > 0) ? TRUE : FALSE;

    //EN Person ever parent
    logical is_ever_parent = (entrances(lives_with_dependent_child,TRUE) > 0) ;

    //EN Lifetime spent single, no dependent children at home
    real time_is_single_alone = duration(is_single_alone, TRUE);

    //EN Lifetime spent single, with dependent children at home
    real time_is_single_child = duration(is_single_child, TRUE);
```

```
    //EN Lifetime spent in partnership, no dependent children at home
    real time_is_couple_alone = duration(is_couple_alone, TRUE);

    //EN Lifetime spent in partnership, with dependent children at home
    real time_is_couple_child = duration(is_couple_child, TRUE);
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Groups
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_FAMILY_TABLES    //EN Family
{
    tabFamilyType,
    tabCohortFamilyExperience20102014,
    tabLivingWithParents
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table Person tabFamilyType                              //EN Population by age and family type
[is_alive && is_resident && is_year10]
{
    year10 *
    sex + *
    educ_fate+ *
    split(integer_age, AGE5_PART) *
    {
        duration(is_single_alone,TRUE) / duration(),    //EN Single, no dependent children at home
decimals=3
        duration(is_single_child,TRUE) / duration(),    //EN Single, dependent children at home
decimals=3
        duration(is_couple_alone,TRUE) / duration(),    //EN In partnership, no dependent children
at home decimals=3
        duration(is_couple_child,TRUE) / duration()     //EN In partnership, dependent children at
home decimals=3
    }
};

table Person tabCohortFamilyExperience20102014          //EN Cohort 2010-2014 family experience
[is_resident && trigger_transitions(is_alive,TRUE,FALSE) && year_of_birth >= 2010 && year_of_birth
< 2015 && person_type==PT_CHILD]
{
    sex + *
    is_ever_parent+ *
    educ_fate+ *
    {
        value_in(time_is_single_alone) / unit,          //EN Average time lived single, no
dependent children at home decimals=3
        value_in(time_is_single_child) / unit,          //EN Average time lived single, dependent
children at home decimals=3
        value_in(time_is_couple_alone) / unit,          //EN Average time lived in partnership, no
dependent children at home decimals=3
        value_in(time_is_couple_child) / unit           //EN Average time lived in partnership,
dependent children at home decimals=3
    }
};

table Person tabLivingWithParents                       //EN Living with parents
[is_resident && in_projected_time && is_year10]
```

```
{
    sex + *
    year10 *
    {
        duration(family_role,FR_CHILD) / duration()     //EN Living with parent(s) decimals=3
    }
    * split(integer_age, AGE_1025)
    * educ_fate +
};
```

The TablesFertility.mpp Module

This module implements a collection of tables related to fertility.

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Groups
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_FERTILITY_TABLES     //EN Fertility
{
    tabFertilityBirths,
    tabFertilityAgeBirth,
    tabFertilityCohortChildlessness,
    tabFertilityMaleCohortChildlesness,
    tabFertilityFemaleChildlessness
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table Person tabFertilityBirths //EN Births
[is_alive && in_projected_time && is_resident && sex==FEMALE &&
WITHIN(FERTILE_AGE_RANGE,integer_age)]
{
    {
        parity,                                                 //EN Births
        transitions(parity,0,1),                                //EN First births
        parity/duration(),                                      //EN Birth rate decimals=4
        transitions(parity,0,1)/duration()                      //EN First birth rate
decimals=4
    }
    * educ_fate+
    * fertile_age+
    * sim_year
};


table Person tabFertilityAgeBirth //EN Age at birth
[is_alive && in_projected_time && is_resident && sex==FEMALE &&
WITHIN(FERTILE_AGE_RANGE,integer_age)]
{
    {
```

```
        value_at_changes(parity,age) / changes(parity),                 //EN Average age birth
decimals=2
        value_at_transitions(parity,0,1,age) / transitions(parity,0,1)  //EN Average age 1st birth
decimals=2
    }
    * educ_fate+
    * sim_year
};

actor Person
{
    YOB_BIRTH1 yob1 = COERCE(YOB_BIRTH1,year_of_birth); //EN Year of birth
};

table Person tabFertilityCohortChildlessness                            //EN Cohort childlessness
[is_alive && integer_age==50 && is_resident && sex==FEMALE && WITHIN(YOB_BIRTH1,year_of_birth)]
{
    {
        duration(parity,0) / duration()                                 //EN Female cohort
Childlessness decimals=2
    }
    * educ_fate+
    * yob1
};

table Person tabFertilityFemaleChildlessness                            //EN Female period
childlessness
[is_alive && is_resident && sex==FEMALE && is_year10]
{
    year10 *
    {
        duration(parity,0) / duration()                                 //EN Cohort Childlessness
decimals=2
    }
    * integer_age
    * educ_fate+
};


table Person tabFertilityMaleCohortChildlesness     //EN Male Cohort Childlessness (at death - run
to 2150!)
[sex == MALE && is_resident && trigger_transitions(is_alive,TRUE,FALSE) &&
WITHIN(YOB_BIRTH1,year_of_birth)]
{
    {
        value_in(never_father) / unit               //EN Childlessenss decimals=2
    }
    * yob1
    * educ_fate +
};
```

The TablesMigration.mpp Module

This module implements table output related to international migration

```
////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
// Table Groups
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_MIGRATION_TABLES  //EN Migration
{
    tabNumberMigrants
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table Person tabNumberMigrants                          //EN Number of migrants
[is_alive && in_projected_time]
{
    sex + *
    {
        //EN Immigrants
        transitions(ever_resident,FALSE,TRUE),

        //EN Emigrants
        transitions(is_resident,TRUE,FALSE),

        //EN Net migration
        transitions(ever_resident,FALSE,TRUE) - transitions(is_resident,TRUE,FALSE)
    }
    * integer_age+
    * sim_year
};
```

The TablesMortality.mpp Module

This module implements a collection of tables related to mortality.

```


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table States
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    logical is_ever_60 = (entrances(integer_age,60) > 0); //EN Ever 60
    logical is_ever_65 = (entrances(integer_age,65) > 0); //EN Ever 65
    logical is_ever_70 = (entrances(integer_age,70) > 0); //EN Ever 70
    logical is_ever_75 = (entrances(integer_age,75) > 0); //EN Ever 75
    logical is_ever_80 = (entrances(integer_age,80) > 0); //EN Ever 80
    logical is_ever_85 = (entrances(integer_age,85) > 0); //EN Ever 85
    logical is_ever_90 = (entrances(integer_age,90) > 0); //EN Ever 90
    logical is_ever_95 = (entrances(integer_age,95) > 0); //EN Ever 95
    logical is_ever_100 = (entrances(integer_age,100) > 0); //EN Ever 100
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
// Table Groups
////////////////////////////////////////////////////////////////////////////////////////////////
//


table_group TG_MORTALITY_TABLES //EN Mortality
{
    tabPeriodMortality,
    tabCohortLifeExpectancy20102014,
    tabMortalityNumberDeaths
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table Person tabCohortLifeExpectancy20102014    //EN Cohort 2010-2014 Life Expectancy
[is_resident && trigger_transitions(is_alive,TRUE,FALSE)
&& year_of_birth >= 2010 && year_of_birth < 2015 && person_type==PT_CHILD]
{
    sex + *
    educ_fate+ *
    {
        value_in(age) / unit,                       //EN Life expectancy decimals=3
        value_in(is_ever_60) / unit,                //EN Proportion surviving until 60 decimals=3
        value_in(is_ever_65) / unit,                //EN Proportion surviving until 65 decimals=3
        value_in(is_ever_70) / unit,                //EN Proportion surviving until 70 decimals=3
        value_in(is_ever_75) / unit,                //EN Proportion surviving until 75 decimals=3
        value_in(is_ever_80) / unit,                //EN Proportion surviving until 80 decimals=3
        value_in(is_ever_85) / unit,                //EN Proportion surviving until 85 decimals=3
        value_in(is_ever_90) / unit,                //EN Proportion surviving until 90 decimals=3
        value_in(is_ever_95) / unit,                //EN Proportion surviving until 95 decimals=3
        value_in(is_ever_100) / unit                //EN Proportion surviving until 100 decimals=3
    }
};

table Person tabPeriodMortality                     //EN Period mortality rates
[is_resident && in_projected_time && is_year10]
{
    sex + *
    year10 *
    {
        transitions(is_alive,TRUE,FALSE) / duration() //EN Mortality rate decimals=3
    }
    * integer_age
    * educ_fate+
};

table Person tabMortalityNumberDeaths                   //EN Number of deaths
[is_resident && in_projected_time]
{
    {
        transitions(is_alive,TRUE,FALSE)               //EN Deaths
    }
    * sim_year
};
```

The TablesNtaBasics.mpp Module

This module implements a set of output tables related to National Transfer Accounts (NTAs). Tables include national totlas for all NTA variables, average age profiles, and measures of the Life Course Deficit.

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Dimensions
////////////////////////////////////////////////////////////////////////////////////////////////
//

partition AGE5105_PART                                          //EN Age Groups
{
    5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,
    65, 70, 75, 80, 85, 90, 95, 100
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table States
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    SIM_YEAR_RANGE sim_year = COERCE(SIM_YEAR_RANGE, calendar_year); //EN Year
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Groups
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_BasicNtaTabs //EN Basic NTA Tables
{
    TabBasicNta, TabBasicNtaByAge, tabLCDperCap, tabPopulationByAgeNta
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table Person TabBasicNta                        //EN Basic NTA totals
[WITHIN(SIM_YEAR_RANGE, calendar_year) && is_resident]
{
    {
        duration(),                             //EN Population
        weighted_duration(base_CFE),            //EN Private Consumption Education (CFE)
        weighted_duration(base_CFH),            //EN Private Consumption Health (CFH)
        weighted_duration(base_CFX),            //EN Private Consumption other than Education and
Health (CFX)
        weighted_duration(base_CGE),            //EN Public Consumption Education (CGE)
        weighted_duration(base_CGH),            //EN Public Consumption Health (CGH)
        weighted_duration(base_CGX),            //EN Public Consumption other than Education and
Health (CGX)
        weighted_duration(base_TGSOAI),         //EN Public Transfers Pensions, Inflows (TGSOAI)
        weighted_duration(base_TGXCI),          //EN Public Transfers Other Cash Inflows (TGXCI)
        weighted_duration(base_TGXII),          //EN Public Transfers Other In-Kind Inflows
(TGXII)
```

```
        weighted_duration(base_TGEI),           //EN Public Transfers Education Inflows (TGEI)
        weighted_duration(base_TGHI),           //EN Public Transfers Health Inflows (TGHI)
        weighted_duration(base_TGO),            //EN Public Transfers Outflows (TGO)
        weighted_duration(base_TFB),            //EN Net Interhousehold Transfers (TFB)
        weighted_duration(base_TFW),            //EN Net Intrahousehold Transfers (TFW)
        weighted_duration(base_SF),             //EN Private Saving (SF)
        weighted_duration(base_SG),             //EN Public Saving (SG)
        weighted_duration(base_YL),             //EN Labor Income (LY)
        weighted_duration(base_YAF),            //EN Private Asset Income (YAF)
        weighted_duration(base_YAG)             //EN Public Asset Income (YAG)
    }
    * sim_year
};

table Person TabBasicNtaByAge                   //EN NTA by age
[WITHIN(SIM_YEAR_RANGE, calendar_year) && is_resident]
{
    sim_year*
    {
        duration(),                                         //EN Population
        weighted_duration(base_CFE) / duration(),           //EN Private Consumption Education
(CFE)
        weighted_duration(base_CFH) / duration(),           //EN Private Consumption Health (CFH)
        weighted_duration(base_CFX) / duration(),           //EN Private Consumption other than
Education and Health (CFX)
        weighted_duration(base_CGE) / duration(),           //EN Public Consumption Education
(CGE)
        weighted_duration(base_CGH) / duration(),           //EN Public Consumption Health (CGH)
        weighted_duration(base_CGX) / duration(),           //EN Public Consumption other than
Education and Health (CGX)
        weighted_duration(base_TGSOAI) / duration(),        //EN Public Transfers Pensions,
Inflows (TGSOAI)
        weighted_duration(base_TGXCI) / duration(),         //EN Public Transfers Other Cash
Inflows (TGXCI)
        weighted_duration(base_TGXII) / duration(),         //EN Public Transfers Other In-Kind
Inflows (TGXII)
        weighted_duration(base_TGEI) / duration(),          //EN Public Transfers Education
Inflows (TGEI)
        weighted_duration(base_TGHI) / duration(),          //EN Public Transfers Health Inflows
(TGHI)
        weighted_duration(base_TGO) / duration(),           //EN Public Transfers Outflows (TGO)
        weighted_duration(base_TFB) / duration(),           //EN Net Interhousehold Transfers
(TFB)
        weighted_duration(base_TFW) / duration(),           //EN Net Intrahousehold Transfers
(TFW)
        weighted_duration(base_SF) / duration(),            //EN Private Saving (SF)
        weighted_duration(base_SG) / duration(),            //EN Public Saving (SG)
        weighted_duration(base_YL) / duration(),            //EN Labor Income (LY)
        weighted_duration(base_YAF) / duration(),           //EN Private Asset Income (YAF)
        weighted_duration(base_YAG) / duration()            //EN Public Asset Income (YAG)
    }
    * integer_age
};

table Person tabLCDperCap                                   //EN Life Course Deficit
[WITHIN(SIM_YEAR_RANGE, calendar_year) && is_resident]
{
    {
        (weighted_duration(base_CFE) + weighted_duration(base_CFH) + weighted_duration(base_CFX) +
weighted_duration(base_CGE) + weighted_duration(base_CGH) + weighted_duration(base_CGX)) /
duration(), //EN average consumption
        weighted_duration(base_YL) / duration(),            //EN Average labor Income (LY)
        -(weighted_duration(base_CFE) + weighted_duration(base_CFH) + weighted_duration(base_CFX)
+ weighted_duration(base_CGE) + weighted_duration(base_CGH) + weighted_duration(base_CGX) -
weighted_duration(base_YL)) / duration() //EN Average gap
    }
```

```
* sim_year
};

table Person tabPopulationByAgeNta                    //EN Population by age group and NTA type
[is_resident && is_year10]
{
    year10 *
    tab_fam_index *
    sex *
    {
        duration()
    }
    * nta_pop_group                      //EN Population Group
    * nta_educ+
};
```

The TablesNtaGenAccounts.mpp Module

This module implements table output related to Full General Accounts (FGAs).

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Groups
////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_FullGenerationalAccounts             //EN NTA Full Generational Accounts
{
    tabFgaUnadjustedTotals,
    tabFgaPV,
    tabFgaPVDetail,
    tabAdjustmentFga,
    tabFgaPV_2010CW,
    tabFgaPV_2040CW,
    tabFgaPV_2010CWdetail,
    tabFgaPV_2040CWdetail
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////
//

table Person tabFgaUnadjustedTotals                 //EN FGA Totals
[is_resident && in_projected_time]
{
    {
        weighted_duration(fga_YL),                  //EN YL total
        weighted_duration(fga_TF),                  //EN TF total
        weighted_duration(fga_TGI),                 //EN TGI total
        weighted_duration(fga_TGO),                 //EN TGO total
        weighted_duration(fga_TF_adj),              //EN TF total adjusted
        weighted_duration(fga_TGI_adj),             //EN TGI total adjusted
        weighted_duration(fga_TGO_adj)              //EN TGO total adjusted
    }
    * sim_year
};

table Person tabFgaPV                               //EN FGA PV
[is_resident && person_type == PT_CHILD
```

```
&& trigger_transitions(is_alive,TRUE,FALSE)]
{
    educ_fate+ *
    {
        value_in(fga_YL_pv) / unit,                 //EN Present Value YL at birth
        value_in(fga_TF_pv) / unit,                 //EN Present Value TF at birth
        value_in(fga_TGI_pv) / unit,                //EN Present Value TGI at birth
        value_in(fga_TGO_pv) / unit,                //EN Present Value TGO at birth
        value_in(fga_TF_adj_pv) / unit,             //EN Present Value TF adjusted at birth
        value_in(fga_TGI_adj_pv) / unit,            //EN Present Value TGI adjsuted at birth
        value_in(fga_TGO_adj_pv) / unit             //EN Present Value TGO adjusted at birth
    }
    * year_of_birth
};

table Person tabFgaPVDetail                         //EN FGA PV by detailed Group
[is_resident && person_type == PT_CHILD && yob_in_2010_2100
&& trigger_transitions(is_alive,TRUE,FALSE)]
{
    educ_fate+ *
    sex+ *
    ever_parent+ *
    {
        value_in(fga_YL_pv) / unit,                 //EN Present Value YL at birth
        value_in(fga_TF_pv) / unit,                 //EN Present Value TF at birth
        value_in(fga_TGI_pv) / unit,                //EN Present Value TGI at birth
        value_in(fga_TGO_pv) / unit,                //EN Present Value TGO at birth
        value_in(fga_TF_adj_pv) / unit,             //EN Present Value TF adjusted at birth
        value_in(fga_TGI_adj_pv) / unit,            //EN Present Value TGI adjsuted at birth
        value_in(fga_TGO_adj_pv) / unit,            //EN Present Value TGO adjusted at birth
        unit                                        //EN Persons
    }
    * yob_2010_2100
};


table Globals tabAdjustmentFga                      //EN FGA Adjustment factors
[globals_in_projected_time]
{
    {
        max_value_in(adj_factor_TFI),               //EN Adjustment TFI decimals=3
        max_value_in(adj_factor_TFO),               //EN Adjustment TFO decimals=3
        max_value_in(adj_factor_TGI),               //EN Adjustment TGI decimals=3
        max_value_in(adj_factor_TGO)                //EN Adjustment TGO decimals=3
    }
    * global_tab_sim_year
};

table Person tabFgaPV_2010CW                          //EN FGA PV 2010
[is_resident && person_type != PT_IMMIGRANT
&& trigger_transitions(is_alive,TRUE,FALSE)
&& year_of_birth <= 2010]
{
    educ_fate+ *
    {
        value_in(fga_YL_pv_2010) / value_in(cohort_weight),           //EN Present Value YL 2010
        value_in(fga_TF_pv_2010) / value_in(cohort_weight),           //EN Present Value TF 2010
        value_in(fga_TGI_pv_2010) / value_in(cohort_weight),          //EN Present Value TGI 2010
        value_in(fga_TGO_pv_2010) / value_in(cohort_weight),          //EN Present Value TGO 2010
        value_in(fga_TF_adj_pv_2010) / value_in(cohort_weight),       //EN Present Value TF
adjusted 2010
        value_in(fga_TGI_adj_pv_2010) / value_in(cohort_weight),      //EN Present Value TGI
adjusted 2010
        value_in(fga_TGO_adj_pv_2010) / value_in(cohort_weight)       //EN Present Value TGO
adjusted 2010
    }
```

```
    * age_2010
};

table Person tabFgaPV_2040CW                          //EN FGA PV 2040
[is_resident && person_type != PT_IMMIGRANT
&& trigger_transitions(is_alive,TRUE,FALSE)
&& year_of_birth <= 2040]
{
    educ_fate+ *
    {
        value_in(fga_YL_pv_2040) / value_in(cohort_weight),           //EN Present Value YL 2040
        value_in(fga_TF_pv_2040) / value_in(cohort_weight),           //EN Present Value TF 2040
        value_in(fga_TGI_pv_2040) / value_in(cohort_weight),          //EN Present Value TGI 2040
        value_in(fga_TGO_pv_2040) / value_in(cohort_weight),          //EN Present Value TGO 2040
        value_in(fga_TF_adj_pv_2040) / value_in(cohort_weight),       //EN Present Value TF
adjusted 2040
        value_in(fga_TGI_adj_pv_2040) / value_in(cohort_weight),      //EN Present Value TGI
adjusted 2040
        value_in(fga_TGO_adj_pv_2040) / value_in(cohort_weight)       //EN Present Value TGO
adjusted 2040
    }
    * age_2040
};

table Person tabFgaPV_2010CWdetail                          //EN FGA PV 2010 - detailed
[is_resident && person_type != PT_IMMIGRANT
&& trigger_transitions(is_alive,TRUE,FALSE)
&& year_of_birth <= 2010]
{
    educ_fate+ *
    sex+ *
    ever_parent+ *
    {
        value_in(fga_YL_pv_2010) / value_in(cohort_weight),           //EN Present Value YL 2010
        value_in(fga_TF_pv_2010) / value_in(cohort_weight),           //EN Present Value TF 2010
        value_in(fga_TGI_pv_2010) / value_in(cohort_weight),          //EN Present Value TGI 2010
        value_in(fga_TGO_pv_2010) / value_in(cohort_weight),          //EN Present Value TGO 2010
        value_in(fga_TF_adj_pv_2010) / value_in(cohort_weight),       //EN Present Value TF
adjusted 2010
        value_in(fga_TGI_adj_pv_2010) / value_in(cohort_weight),      //EN Present Value TGI
adjusted 2010
        value_in(fga_TGO_adj_pv_2010) / value_in(cohort_weight)       //EN Present Value TGO
adjusted 2010
    }
    * age_2010
};

table Person tabFgaPV_2040CWdetail                          //EN FGA PV 2040 - detailed
[is_resident && person_type != PT_IMMIGRANT
&& trigger_transitions(is_alive,TRUE,FALSE)
&& year_of_birth <= 2040]
{
    educ_fate+ *
    sex+ *
    ever_parent+ *
    {
        value_in(fga_YL_pv_2040) / value_in(cohort_weight),           //EN Present Value YL 2040
        value_in(fga_TF_pv_2040) / value_in(cohort_weight),           //EN Present Value TF 2040
        value_in(fga_TGI_pv_2040) / value_in(cohort_weight),          //EN Present Value TGI 2040
        value_in(fga_TGO_pv_2040) / value_in(cohort_weight),          //EN Present Value TGO 2040
        value_in(fga_TF_adj_pv_2040) / value_in(cohort_weight),       //EN Present Value TF
adjusted 2040
        value_in(fga_TGI_adj_pv_2040) / value_in(cohort_weight),      //EN Present Value TGI
adjusted 2040
        value_in(fga_TGO_adj_pv_2040) / value_in(cohort_weight)       //EN Present Value TGO
adjusted 2040
```

```
    }
    * age_2040
};
```

The TablesNtaIndicators.mpp Module

This modules produces output tables related to NTA indicators, including the Support Ration, the Imapct Index, and aggregate measures of effective labour, consumers, income, capital, as well as wages in dthe interest rate in a closed economy.

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Groups
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_NTAIndicators                    //EN NTA Indicators and Impact Index Lee 2017
{
    TabNTAIndicators
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table State TabNTAIndicators                    //EN NTA Indicators and Impact Index Lee 2017
[WITHIN(SIM_YEAR_RANGE,state_year) && nta_is_updated]
{
    {
        max_value_in(nta_current_SR),           //EN Support Ratio SR decimals=4
        max_value_in(nta_current_closed_IMP),   //EN Impact Index IMP closed economy decimals=4
        max_value_in(nta_current_open_IMP),     //EN Impact Index IMP open economy decimals=4
        max_value_in(nta_current_N),            //EN Current reference consumers N
        max_value_in(nta_current_L),            //EN Current total Labor L
        max_value_in(nta_current_C),            //EN Current total Consumption
        max_value_in(nta_current_Y),            //EN Current total Income Y
        max_value_in(nta_current_K),            //EN Current total Capital K
        max_value_in(nta_current_w),            //EN Current wage w closed economy decimals=4
        max_value_in(nta_current_r)             //EN Current interest rate r closed economy
decimals=4
    }
    * tab_state_year
};
```

The TablesNtaValidation.mpp Module

The NTA Validation Module is an optional module introducing a set of validation tables comparing the aggregate outcome of NTA variables by education and family type with

published aggregates by sex a/o age. For each of the 19 NTA variables, a table is produced comparing aggregated simulated values in the starting year with published NTA values and providing aggregate simulated outcomes by education group. Also a table of the population composition by all considered dimensions (age, sex, education, family type) is produced. This module can be removed to save memory space.

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// Dimensions
////////////////////////////////////////////////////////////////////////////////////////////
//

range TAB_FAM_INDEX { 0,5 }; //EN Family Index

////////////////////////////////////////////////////////////////////////////////////////////
//
// Actor states
////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    //EN Family Index
    TAB_FAM_INDEX tab_fam_index = (nta_pop_group == NPG_OLD) ?
        COERCE(TAB_FAM_INDEX, nta_old_index) : COERCE(TAB_FAM_INDEX, nta_fam_index);

    // NTA by age only
    double age_CFE = NtaAge[integer_age][CFE];              //EN CFE by age
    double age_CFH = NtaAge[integer_age][CFH];              //EN CFH by age
    double age_CFX = NtaAge[integer_age][CFX];              //EN CFX by age
    double age_CGE = NtaAge[integer_age][CGE];              //EN CGE by age
    double age_CGH = NtaAge[integer_age][CGH];              //EN CGH by age
    double age_CGX = NtaAge[integer_age][CGX];              //EN CGX by age
    double age_TGSOAI = NtaAge[integer_age][TGSOAI];        //EN TGSOAI by age
    double age_TGXCI = NtaAge[integer_age][TGXCI];          //EN TGXCI by age
    double age_TGXII = NtaAge[integer_age][TGXII];          //EN TGXII by age
    double age_TGEI = NtaAge[integer_age][TGEI];            //EN TGEI by age
    double age_TGHI = NtaAge[integer_age][TGHI];            //EN TGHI by age
    double age_TGO = NtaAge[integer_age][TGO];              //EN TGO by age
    double age_TFB = NtaAge[integer_age][TFB];              //EN TFB by age
    double age_TFW = NtaAge[integer_age][TFW];              //EN TFW by age
    double age_SF = NtaAge[integer_age][SF];                //EN SF by age
    double age_SG = NtaAge[integer_age][SG];                //EN SG by age
    double age_YL = NtaAge[integer_age][YL];                //EN LY by age
    double age_YAF = NtaAge[integer_age][YAF];              //EN YAF by age
    double age_YAG = NtaAge[integer_age][YAG];              //EN YAG by age

    // NTA by age and sex
    double sex_CFE = NtaSex[sex][integer_age][CFE];         //EN CFE by age and sex
    double sex_CFH = NtaSex[sex][integer_age][CFH];         //EN CFH by age and sex
    double sex_CFX = NtaSex[sex][integer_age][CFX];         //EN CFX by age and sex
    double sex_CGE = NtaSex[sex][integer_age][CGE];         //EN CGE by age and sex
    double sex_CGH = NtaSex[sex][integer_age][CGH];         //EN CGH by age and sex
    double sex_CGX = NtaSex[sex][integer_age][CGX];         //EN CGX by age and sex
    double sex_TGSOAI = NtaSex[sex][integer_age][TGSOAI];   //EN TGSOAI by age and sex
    double sex_TGXCI = NtaSex[sex][integer_age][TGXCI];     //EN TGXCI by age and sex
    double sex_TGXII = NtaSex[sex][integer_age][TGXII];     //EN TGXII by age and sex
    double sex_TGEI = NtaSex[sex][integer_age][TGEI];       //EN TGEI by age and sex
    double sex_TGHI = NtaSex[sex][integer_age][TGHI];       //EN TGHI by age and sex
```

```
    double sex_TGO = NtaSex[sex][integer_age][TGO];         //EN TGO by age and sex
    double sex_TFB = NtaSex[sex][integer_age][TFB];         //EN TFB by age and sex
    double sex_TFW = NtaSex[sex][integer_age][TFW];         //EN TFW by age and sex
    double sex_SF = NtaSex[sex][integer_age][SF];           //EN SF by age and sex
    double sex_SG = NtaSex[sex][integer_age][SG];           //EN SG by age  and sex
    double sex_YL = NtaSex[sex][integer_age][YL];           //EN LY by age and sex
    double sex_YAF = NtaSex[sex][integer_age][YAF];         //EN YAF by age and sex
    double sex_YAG = NtaSex[sex][integer_age][YAG];         //EN YAG by age and sex
};

////////////////////////////////////////////////////////////////////////////////////////////
//
// Validation Tables
////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_NTA_ValidationTables     //EN NTA Validation Tables
{
    TabNtaValidation_CFE, TabNtaValidation_CFH, TabNtaValidation_CFX, TabNtaValidation_CGE,
    TabNtaValidation_CGH, TabNtaValidation_CGX, TabNtaValidation_TGSOAI, TabNtaValidation_TGXCI,
    TabNtaValidation_TGXII, TabNtaValidation_TGEI, TabNtaValidation_TGHI, TabNtaValidation_TGO,
    TabNtaValidation_TFB, TabNtaValidation_TFW, TabNtaValidation_SF, TabNtaValidation_SG,
    TabNtaValidation_YL, TabNtaValidation_YAF, TabNtaValidation_YAG, TabNtaPopulation2010
};

table Person TabNtaValidation_CFE                                                   //EN CFE
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CFE) / duration(),                                   //EN CFE
        weighted_duration(sex,FEMALE, base_CFE) / duration(sex,FEMALE),             //EN CFE female
        weighted_duration(sex,MALE, base_CFE) / duration(sex,MALE),                 //EN CFE male

        weighted_duration(nta_educ,NE_LOW, base_CFE) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_CFE) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_CFE) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_CFE) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_CFE) / duration(sex,FEMALE),           //EN Agenta CFE
female
        weighted_duration(sex,MALE,sex_CFE) / duration(sex,MALE),               //EN Agenta CFE
male
        weighted_duration(age_CFE) / duration()                                 //EN Agenta CFE by
age
    }
    * integer_age
};

table Person TabNtaValidation_CFH                                                   //EN CFH
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CFH) / duration(),                                   //EN CFH
        weighted_duration(sex,FEMALE, base_CFH) / duration(sex,FEMALE),             //EN CFH female
        weighted_duration(sex,MALE, base_CFH) / duration(sex,MALE),                 //EN CFH male

        weighted_duration(nta_educ,NE_LOW, base_CFH) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_CFH) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
```

```
        weighted_duration(nta_educ,NE_HIGH, base_CFH) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_CFH) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_CFH) / duration(sex,FEMALE),           //EN Agenta CFH
female
        weighted_duration(sex,MALE,sex_CFH) / duration(sex,MALE),               //EN Agenta CFH
male
        weighted_duration(age_CFH) / duration()                                 //EN Agenta CFH by
age
    }
    * integer_age
};

table Person TabNtaValidation_CFX                                               //EN CFX
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CFX) / duration(),                               //EN CFX
        weighted_duration(sex,FEMALE, base_CFX) / duration(sex,FEMALE),         //EN CFX female
        weighted_duration(sex,MALE, base_CFX) / duration(sex,MALE),             //EN CFX male

        weighted_duration(nta_educ,NE_LOW, base_CFX) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_CFX) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_CFX) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_CFX) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_CFX) / duration(sex,FEMALE),           //EN Agenta CFX
female
        weighted_duration(sex,MALE,sex_CFX) / duration(sex,MALE),               //EN Agenta CFX
male
        weighted_duration(age_CFX) / duration()                                 //EN Agenta CFX by
age
    }
    * integer_age
};

table Person TabNtaValidation_CGE                                               //EN CGE
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CGE) / duration(),                               //EN CGE
        weighted_duration(sex,FEMALE, base_CGE) / duration(sex,FEMALE),         //EN CGE female
        weighted_duration(sex,MALE, base_CGE) / duration(sex,MALE),             //EN CGE male

        weighted_duration(nta_educ,NE_LOW, base_CGE) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_CGE) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_CGE) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_CGE) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_CGE) / duration(sex,FEMALE),           //EN Agenta CGE
female
        weighted_duration(sex,MALE,sex_CGE) / duration(sex,MALE),               //EN Agenta CGE
male
```

```
        weighted_duration(age_CGE) / duration()                                 //EN Agenta CGE by
age
    }
    * integer_age
};


table Person TabNtaValidation_CGH                                               //EN CGH
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CGH) / duration(),                               //EN CGH
        weighted_duration(sex,FEMALE, base_CGH) / duration(sex,FEMALE),         //EN CGH female
        weighted_duration(sex,MALE, base_CGH) / duration(sex,MALE),             //EN CGH male

        weighted_duration(nta_educ,NE_LOW, base_CGH) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_CGH) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_CGH) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_CGH) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_CGH) / duration(sex,FEMALE),           //EN Agenta CGH
female
        weighted_duration(sex,MALE,sex_CGH) / duration(sex,MALE),               //EN Agenta CGH
male
        weighted_duration(age_CGH) / duration()                                 //EN Agenta CGH by
age
    }
    * integer_age
};


table Person TabNtaValidation_CGX                                               //EN CGX
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_CGX) / duration(),                               //EN CGX
        weighted_duration(sex,FEMALE, base_CGX) / duration(sex,FEMALE),         //EN CGX female
        weighted_duration(sex,MALE, base_CGX) / duration(sex,MALE),             //EN CGX male

        weighted_duration(nta_educ,NE_LOW, base_CGX) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_CGX) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_CGX) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_CGX) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_CGX) / duration(sex,FEMALE),           //EN Agenta CGX
female
        weighted_duration(sex,MALE,sex_CGX) / duration(sex,MALE),               //EN Agenta CGX
male
        weighted_duration(age_CGX) / duration()                                 //EN Agenta CGX by
age
    }
    * integer_age
};

table Person TabNtaValidation_TGSOAI                                              //EN TGSOAI
Validation
```

```
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGSOAI) / duration(),                                    //EN TGSOAI
        weighted_duration(sex,FEMALE, base_TGSOAI) / duration(sex,FEMALE),          //EN TGSOAI
female
        weighted_duration(sex,MALE, base_TGSOAI) / duration(sex,MALE),              //EN TGSOAI
male

        weighted_duration(nta_educ,NE_LOW, base_TGSOAI) / duration(nta_educ,NE_LOW),        //EN
CFE educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_TGSOAI) / duration(nta_educ,NE_MEDIUM),  //EN
CFE educ medium
        weighted_duration(nta_educ,NE_HIGH, base_TGSOAI) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_TGSOAI) / duration(nta_educ,NE_NN),    //EN CFE
educ nn

        weighted_duration(sex,FEMALE,sex_TGSOAI) / duration(sex,FEMALE),            //EN Agenta
TGSOAI female
        weighted_duration(sex,MALE,sex_TGSOAI) / duration(sex,MALE),                //EN Agenta
TGSOAI male
        weighted_duration(age_TGSOAI) / duration()                                      //EN Agenta
TGSOAI by age
    }
    * integer_age
};

table Person TabNtaValidation_TGXCI                                                 //EN TGXCI
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGXCI) / duration(),                                     //EN TGXCI
        weighted_duration(sex,FEMALE, base_TGXCI) / duration(sex,FEMALE),           //EN TGXCI
female
        weighted_duration(sex,MALE, base_TGXCI) / duration(sex,MALE),               //EN TGXCI male

        weighted_duration(nta_educ,NE_LOW, base_TGXCI) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_TGXCI) / duration(nta_educ,NE_MEDIUM),  //EN
CFE educ medium
        weighted_duration(nta_educ,NE_HIGH, base_TGXCI) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_TGXCI) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_TGXCI) / duration(sex,FEMALE),             //EN Agenta
TGXCI female
        weighted_duration(sex,MALE,sex_TGXCI) / duration(sex,MALE),                 //EN Agenta
TGXCI male
        weighted_duration(age_TGXCI) / duration()                                       //EN Agenta
TGXCI by age
    }
    * integer_age
};

table Person TabNtaValidation_TGXII                                                 //EN TGXII
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGXII) / duration(),                                     //EN TGXII
        weighted_duration(sex,FEMALE, base_TGXII) / duration(sex,FEMALE),           //EN TGXII
female
        weighted_duration(sex,MALE, base_TGXII) / duration(sex,MALE),               //EN TGXII male
```

```
        weighted_duration(nta_educ,NE_LOW, base_TGXII) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_TGXII) / duration(nta_educ,NE_MEDIUM),  //EN
CFE educ medium
        weighted_duration(nta_educ,NE_HIGH, base_TGXII) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_TGXII) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_TGXII) / duration(sex,FEMALE),           //EN Agenta
TGXII female
        weighted_duration(sex,MALE,sex_TGXII) / duration(sex,MALE),               //EN Agenta
TGXII male
        weighted_duration(age_TGXII) / duration()                                 //EN Agenta
TGXII by age
    }
    * integer_age
};

table Person TabNtaValidation_TGEI                                              //EN TGEI
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGEI) / duration(),                              //EN TGEI
        weighted_duration(sex,FEMALE, base_TGEI) / duration(sex,FEMALE),        //EN TGEI female
        weighted_duration(sex,MALE, base_TGEI) / duration(sex,MALE),            //EN TGEI male

        weighted_duration(nta_educ,NE_LOW, base_TGEI) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_TGEI) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_TGEI) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_TGEI) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_TGEI) / duration(sex,FEMALE),          //EN Agenta TGEI
female
        weighted_duration(sex,MALE,sex_TGEI) / duration(sex,MALE),              //EN Agenta TGEI
male
        weighted_duration(age_TGEI) / duration()                                //EN Agenta TGEI
by age
    }
    * integer_age
};

table Person TabNtaValidation_TGHI                                              //EN TGHI
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGHI) / duration(),                              //EN TGHI
        weighted_duration(sex,FEMALE, base_TGHI) / duration(sex,FEMALE),        //EN TGHI female
        weighted_duration(sex,MALE, base_TGHI) / duration(sex,MALE),            //EN TGHI male

        weighted_duration(nta_educ,NE_LOW, base_TGHI) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_TGHI) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_TGHI) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_TGHI) / duration(nta_educ,NE_NN),    //EN CFE educ
nn
```

```
        weighted_duration(sex,FEMALE,sex_TGHI) / duration(sex,FEMALE),           //EN Agenta TGHI
female
        weighted_duration(sex,MALE,sex_TGHI) / duration(sex,MALE),               //EN Agenta TGHI
male
        weighted_duration(age_TGHI) / duration()                                 //EN Agenta TGHI
by age
    }
    * integer_age
};

table Person TabNtaValidation_TGO                                               //EN TGO
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TGO) / duration(),                                //EN TGO
        weighted_duration(sex,FEMALE, base_TGO) / duration(sex,FEMALE),          //EN TGO female
        weighted_duration(sex,MALE, base_TGO) / duration(sex,MALE),              //EN TGO male

        weighted_duration(nta_educ,NE_LOW, base_TGO) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_TGO) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_TGO) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_TGO) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_TGO) / duration(sex,FEMALE),            //EN Agenta TGO
female
        weighted_duration(sex,MALE,sex_TGO) / duration(sex,MALE),                //EN Agenta TGO
male
        weighted_duration(age_TGO) / duration()                                  //EN Agenta TGO by
age
    }
    * integer_age
};

table Person TabNtaValidation_TFB                                               //EN TFB
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TFB) / duration(),                                //EN TFB
        weighted_duration(sex,FEMALE, base_TFB) / duration(sex,FEMALE),          //EN TFB female
        weighted_duration(sex,MALE, base_TFB) / duration(sex,MALE),              //EN TFB male

        weighted_duration(nta_educ,NE_LOW, base_TFB) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_TFB) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_TFB) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_TFB) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_TFB) / duration(sex,FEMALE),            //EN Agenta TFB
female
        weighted_duration(sex,MALE,sex_TFB) / duration(sex,MALE),                //EN Agenta TFB
male
        weighted_duration(age_TFB) / duration()                                  //EN Agenta TFB by
age
    }
    * integer_age
};
```

```
table Person TabNtaValidation_TFW                                          //EN TFW
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_TFW) / duration(),                          //EN TFW
        weighted_duration(sex,FEMALE, base_TFW) / duration(sex,FEMALE),    //EN TFW female
        weighted_duration(sex,MALE, base_TFW) / duration(sex,MALE),        //EN TFW male

        weighted_duration(nta_educ,NE_LOW, base_TFW) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_TFW) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_TFW) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_TFW) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_TFW) / duration(sex,FEMALE),      //EN Agenta TFW
female
        weighted_duration(sex,MALE,sex_TFW) / duration(sex,MALE),          //EN Agenta TFW
male
        weighted_duration(age_TFW) / duration()                            //EN Agenta TFW by
age
    }
    * integer_age
};

table Person TabNtaValidation_SF                                          //EN SF Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_SF) / duration(),                           //EN SF
        weighted_duration(sex,FEMALE, base_SF) / duration(sex,FEMALE),     //EN SF female
        weighted_duration(sex,MALE, base_SF) / duration(sex,MALE),         //EN SF male

        weighted_duration(nta_educ,NE_LOW, base_SF) / duration(nta_educ,NE_LOW),     //EN CFE educ
low
        weighted_duration(nta_educ,NE_MEDIUM, base_SF) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_SF) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_SF) / duration(nta_educ,NE_NN),    //EN CFE educ nn

        weighted_duration(sex,FEMALE,sex_SF) / duration(sex,FEMALE),       //EN Agenta SF
female
        weighted_duration(sex,MALE,sex_SF) / duration(sex,MALE),           //EN Agenta SF male
        weighted_duration(age_SF) / duration()                             //EN Agenta SF by
age
    }
    * integer_age
};

table Person TabNtaValidation_SG                                          //EN SG Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_SG) / duration(),                           //EN SG
        weighted_duration(sex,FEMALE, base_SG) / duration(sex,FEMALE),     //EN SG female
        weighted_duration(sex,MALE, base_SG) / duration(sex,MALE),         //EN SG male

        weighted_duration(nta_educ,NE_LOW, base_SG) / duration(nta_educ,NE_LOW),     //EN CFE educ
low
        weighted_duration(nta_educ,NE_MEDIUM, base_SG) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
```

```
        weighted_duration(nta_educ,NE_HIGH, base_SG) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_SG) / duration(nta_educ,NE_NN),    //EN CFE educ nn

        weighted_duration(sex,FEMALE,sex_SG) / duration(sex,FEMALE),           //EN Agenta SG
female
        weighted_duration(sex,MALE,sex_SG) / duration(sex,MALE),               //EN Agenta SG male
        weighted_duration(age_SG) / duration()                                 //EN Agenta SG by
age
    }
    * integer_age
};

table Person TabNtaValidation_YL                                               //EN YL Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_YL) / duration(),                               //EN YL
        weighted_duration(sex,FEMALE, base_YL) / duration(sex,FEMALE),         //EN YL female
        weighted_duration(sex,MALE, base_YL) / duration(sex,MALE),             //EN YL male

        weighted_duration(nta_educ,NE_LOW, base_YL) / duration(nta_educ,NE_LOW),     //EN CFE educ
low
        weighted_duration(nta_educ,NE_MEDIUM, base_YL) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_YL) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_YL) / duration(nta_educ,NE_NN),    //EN CFE educ nn

        weighted_duration(sex,FEMALE,sex_YL) / duration(sex,FEMALE),           //EN Agenta YL
female
        weighted_duration(sex,MALE,sex_YL) / duration(sex,MALE),               //EN Agenta YL male
        weighted_duration(age_YL) / duration()                                 //EN Agenta YL by
age
    }
    * integer_age
};

table Person TabNtaValidation_YAF                                               //EN YAF
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_YAF) / duration(),                              //EN YAF
        weighted_duration(sex,FEMALE, base_YAF) / duration(sex,FEMALE),        //EN YAF female
        weighted_duration(sex,MALE, base_YAF) / duration(sex,MALE),            //EN YAF male

        weighted_duration(nta_educ,NE_LOW, base_YAF) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_YAF) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_YAF) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_YAF) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_YAF) / duration(sex,FEMALE),          //EN Agenta YAF
female
        weighted_duration(sex,MALE,sex_YAF) / duration(sex,MALE),              //EN Agenta YAF
male
        weighted_duration(age_YAF) / duration()                                //EN Agenta YAF by
age
    }
    * integer_age
};
```

```
table Person TabNtaValidation_YAG                                                    //EN YAG
Validation
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    {
        weighted_duration(base_YAG) / duration(),                                    //EN YAG
        weighted_duration(sex,FEMALE, base_YAG) / duration(sex,FEMALE),              //EN YAG female
        weighted_duration(sex,MALE, base_YAG) / duration(sex,MALE),                  //EN YAG male

        weighted_duration(nta_educ,NE_LOW, base_YAG) / duration(nta_educ,NE_LOW),     //EN CFE
educ low
        weighted_duration(nta_educ,NE_MEDIUM, base_YAG) / duration(nta_educ,NE_MEDIUM),  //EN CFE
educ medium
        weighted_duration(nta_educ,NE_HIGH, base_YAG) / duration(nta_educ,NE_HIGH),    //EN CFE
educ high
        weighted_duration(nta_educ,NE_NN, base_YAG) / duration(nta_educ,NE_NN),    //EN CFE educ
nn

        weighted_duration(sex,FEMALE,sex_YAG) / duration(sex,FEMALE),            //EN Agenta YAG
female
        weighted_duration(sex,MALE,sex_YAG) / duration(sex,MALE),                //EN Agenta YAG
male
        weighted_duration(age_YAG) / duration()                                  //EN Agenta YAG by
age
    }
    * integer_age
};


table Person TabNtaPopulation2010                          //EN NTA POPULATION
[calendar_year == MIN(SIM_YEAR_RANGE) && is_resident]
{
    nta_pop_group *
    tab_fam_index + *
    sex + *
    {
        duration() //EN Duration
    }
    * integer_age
    * nta_educ +
};
```

The TablesNtaVisualisation.mpp Module

This module implements table output as base of the online NTA visualisation tool. The module is optional and can be removed from the application. Table output is produced for two points in time - the starting year and the calendar year 50 years later. Currently, the output for the future is a dummy output only.

```
////////////////////////////////////////////////////////////////////////////////////////
//
// Dimensions
////////////////////////////////////////////////////////////////////////////////////////
//
```

```
classification NTA_VISULAL_YEAR         //EN Year
{
    NVY_BASE,                           //EN Base year
    NVY_BASEP50                         //EN Projected year
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// States used in the table output
////////////////////////////////////////////////////////////////////////////////////////////////
//

actor Person
{
    double current_CFE =  (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_CFE : base_CFE * 1.5;
//EN Private Consumption Education (CFE)
    double current_CFH =  (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_CFH : base_CFH * 1.5;
//EN Private Consumption Health (CFH)
    double current_CFX = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_CFX : base_CFX * 1.5;
//EN Private Consumption other than Education and Health (CFX)
    double current_CGE =  (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_CGE : base_CGE * 1.5;
//EN Public Consumption Education (CGE)
    double current_CGH = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_CGH : base_CGH * 1.5;
//EN Public Consumption Health (CGH)
    double current_CGX = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_CGX : base_CGX * 1.5;
//EN Public Consumption other than Education and Health (CGX)
    double current_TGSOAI = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_TGSOAI : base_TGSOAI *
1.5;   //EN Public Transfers Pensions, Inflows (TGSOAI)
    double current_TGXCI = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_TGXCI : base_TGXCI * 1.5;
//EN Public Transfers Other Cash Inflows (TGXCI)
    double current_TGXII = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_TGXII : base_TGXII * 1.5;
//EN Public Transfers Other In-Kind Inflows (TGXII)
    double current_TGEI = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_TGEI : base_TGEI * 1.5;
//EN Public Transfers Education Inflows (TGEI)
    double current_TGHI = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_TGHI : base_TGHI * 1.5;
//EN Public Transfers Health Inflows (TGHI)
    double current_TGO = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_TGO : base_TGO * 1.5;
//EN Public Transfers Outflows (TGO)
    double current_TFB = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_TFB : base_TFB * 1.5;
//EN Net Interhousehold Transfers (TFB)
    double current_TFW = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_TFW : base_TFW * 1.5;//EN
Net Intrahousehold Transfers (TFW)
    double current_SF = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_SF : base_SF * 1.5;
//EN Private Saving (SF)
    double current_SG = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_SG : base_SG * 1.5;
//EN Public Saving (SG)
    double current_YL = (calendar_year == MIN(SIM_YEAR_RANGE)) ? base_YL : base_YL * 1.5;
//EN Labor Income (LY)
    double current_YAF = (calendar_year == MIN(SIM_YEAR_RANGE)) ?  base_YAF : base_YAF * 1.5;
//EN Private Asset Income (YAF)
    double current_YAG = (calendar_year == MIN(SIM_YEAR_RANGE)) ?  base_YAG : base_YAG * 1.5;
//EN Public Asset Income (YAG)

    logical in_nta_visualisation_year = (calendar_year == MIN(SIM_YEAR_RANGE) || calendar_year ==
MIN(SIM_YEAR_RANGE) + 50);
    NTA_VISULAL_YEAR nta_visual_year = (calendar_year == MIN(SIM_YEAR_RANGE)) ? NVY_BASE :
NVY_BASEP50;     //EN Year
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Groups
////////////////////////////////////////////////////////////////////////////////////////////////
//
```

```
table_group TG_NTA_VISAL //EN NTA Visualisations
{
    tabNtaVisualisationChild,
    tabNtaVisualisationStudent,
    tabNtaVisualisationYoung,
    tabNtaVisualisationAdults,
    tabNtaVisualisationOld
};


////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table Person tabNtaVisualisationChild       //EN NTA Visualisation Children
[ in_nta_visualisation_year && nta_pop_group == NPG_CHILD && is_resident]
{
    nta_visual_year*
    {
        duration(), //EN Population
        weighted_duration(current_CFE) / duration(),    //EN Private Consumption Education (CFE)
        weighted_duration(current_CFH) / duration(),    //EN Private Consumption Health (CFH)
        weighted_duration(current_CFX) / duration(),    //EN Private Consumption other than
Education and Health (CFX)
        weighted_duration(current_CGE) / duration(),    //EN Public Consumption Education (CGE)
        weighted_duration(current_CGH) / duration(),    //EN Public Consumption Health (CGH)
        weighted_duration(current_CGX) / duration(),    //EN Public Consumption other than
Education and Health (CGX)
        weighted_duration(current_TGSOAI) / duration(), //EN Public Transfers Pensions, Inflows
(TGSOAI)
        weighted_duration(current_TGXCI) / duration(),  //EN Public Transfers Other Cash Inflows
(TGXCI)
        weighted_duration(current_TGXII) / duration(),  //EN Public Transfers Other In-Kind
Inflows (TGXII)
        weighted_duration(current_TGEI) / duration(),   //EN Public Transfers Education Inflows
(TGEI)
        weighted_duration(current_TGHI) / duration(),   //EN Public Transfers Health Inflows
(TGHI)
        weighted_duration(current_TGO) / duration(),    //EN Public Transfers Outflows (TGO)
        weighted_duration(current_TFB) / duration(),    //EN Net Interhousehold Transfers (TFB)
        weighted_duration(current_TFW) / duration(),    //EN Net Intrahousehold Transfers (TFW)
        weighted_duration(current_SF) / duration(),     //EN Private Saving (SF)
        weighted_duration(current_SG) / duration(),     //EN Public Saving (SG)
        weighted_duration(current_YL) / duration(),     //EN Labor Income (LY)
        weighted_duration(current_YAF) / duration(),    //EN Private Asset Income (YAF)
        weighted_duration(current_YAG) / duration()     //EN Public Asset Income (YAG)
    }
    *educ_parents +
};


table Person tabNtaVisualisationStudent       //EN NTA Visualisation Young Students
[ in_nta_visualisation_year && nta_pop_group == NPG_YOUNG_STUDENT  && is_resident]
{
    nta_visual_year*
    {
        duration(),                                     //EN Population
        duration(sex,FEMALE) / duration(),              //EN Proportion Female
        weighted_duration(current_CFE) / duration(),    //EN Private Consumption Education (CFE)
        weighted_duration(current_CFH) / duration(),    //EN Private Consumption Health (CFH)
        weighted_duration(current_CFX) / duration(),    //EN Private Consumption other than
Education and Health (CFX)
        weighted_duration(current_CGE) / duration(),    //EN Public Consumption Education (CGE)
        weighted_duration(current_CGH) / duration(),    //EN Public Consumption Health (CGH)
```

```
        weighted_duration(current_CGX) / duration(),    //EN Public Consumption other than
Education and Health (CGX)
        weighted_duration(current_TGSOAI) / duration(), //EN Public Transfers Pensions, Inflows
(TGSOAI)
        weighted_duration(current_TGXCI) / duration(),  //EN Public Transfers Other Cash Inflows
(TGXCI)
        weighted_duration(current_TGXII) / duration(),  //EN Public Transfers Other In-Kind
Inflows (TGXII)
        weighted_duration(current_TGEI) / duration(),   //EN Public Transfers Education Inflows
(TGEI)
        weighted_duration(current_TGHI) / duration(),   //EN Public Transfers Health Inflows
(TGHI)
        weighted_duration(current_TGO) / duration(),    //EN Public Transfers Outflows (TGO)
        weighted_duration(current_TFB) / duration(),    //EN Net Interhousehold Transfers (TFB)
        weighted_duration(current_TFW) / duration(),    //EN Net Intrahousehold Transfers (TFW)
        weighted_duration(current_SF) / duration(),     //EN Private Saving (SF)
        weighted_duration(current_SG) / duration(),     //EN Public Saving (SG)
        weighted_duration(current_YL) / duration(),     //EN Labor Income (LY)
        weighted_duration(current_YAF) / duration(),    //EN Private Asset Income (YAF)
        weighted_duration(current_YAG) / duration()     //EN Public Asset Income (YAG)
    }
    * nta_educ +
};

table Person tabNtaVisualisationYoung       //EN NTA Visualisation Young Workers
[ in_nta_visualisation_year && nta_pop_group == NPG_YOUNG_WORK  && is_resident]
{
    nta_visual_year*
    sex+ *
    nta_family_active+ *
    {
        duration(), //EN Population
        weighted_duration(current_CFE) / duration(),    //EN Private Consumption Education (CFE)
        weighted_duration(current_CFH) / duration(),    //EN Private Consumption Health (CFH)
        weighted_duration(current_CFX) / duration(),    //EN Private Consumption other than
Education and Health (CFX)
        weighted_duration(current_CGE) / duration(),    //EN Public Consumption Education (CGE)
        weighted_duration(current_CGH) / duration(),    //EN Public Consumption Health (CGH)
        weighted_duration(current_CGX) / duration(),    //EN Public Consumption other than
Education and Health (CGX)
        weighted_duration(current_TGSOAI) / duration(), //EN Public Transfers Pensions, Inflows
(TGSOAI)
        weighted_duration(current_TGXCI) / duration(),  //EN Public Transfers Other Cash Inflows
(TGXCI)
        weighted_duration(current_TGXII) / duration(),  //EN Public Transfers Other In-Kind
Inflows (TGXII)
        weighted_duration(current_TGEI) / duration(),   //EN Public Transfers Education Inflows
(TGEI)
        weighted_duration(current_TGHI) / duration(),   //EN Public Transfers Health Inflows
(TGHI)
        weighted_duration(current_TGO) / duration(),    //EN Public Transfers Outflows (TGO)
        weighted_duration(current_TFB) / duration(),    //EN Net Interhousehold Transfers (TFB)
        weighted_duration(current_TFW) / duration(),    //EN Net Intrahousehold Transfers (TFW)
        weighted_duration(current_SF) / duration(),     //EN Private Saving (SF)
        weighted_duration(current_SG) / duration(),     //EN Public Saving (SG)
        weighted_duration(current_YL) / duration(),     //EN Labor Income (LY)
        weighted_duration(current_YAF) / duration(),    //EN Private Asset Income (YAF)
        weighted_duration(current_YAG) / duration()     //EN Public Asset Income (YAG)
    }
    * educ_fate +
};

table Person tabNtaVisualisationAdults       //EN NTA Visualisation Adult Workers
[ in_nta_visualisation_year && nta_pop_group == NPG_ADULT_WORK  && is_resident]
{
    nta_visual_year*
```

```
    sex+ *
    nta_family_active+ *
    {
        duration(), //EN Population
        weighted_duration(current_CFE) / duration(),    //EN Private Consumption Education (CFE)
        weighted_duration(current_CFH) / duration(),    //EN Private Consumption Health (CFH)
        weighted_duration(current_CFX) / duration(),    //EN Private Consumption other than
Education and Health (CFX)
        weighted_duration(current_CGE) / duration(),    //EN Public Consumption  Education (CGE)
        weighted_duration(current_CGH) / duration(),    //EN Public Consumption Health (CGH)
        weighted_duration(current_CGX) / duration(),    //EN Public Consumption other than
Education and Health (CGX)
        weighted_duration(current_TGSOAI) / duration(), //EN Public Transfers Pensions, Inflows
(TGSOAI)
        weighted_duration(current_TGXCI) / duration(),  //EN Public Transfers Other Cash Inflows
(TGXCI)
        weighted_duration(current_TGXII) / duration(),  //EN Public Transfers Other In-Kind
Inflows (TGXII)
        weighted_duration(current_TGEI) / duration(),   //EN Public Transfers Education Inflows
(TGEI)
        weighted_duration(current_TGHI) / duration(),   //EN Public Transfers Health Inflows
(TGHI)
        weighted_duration(current_TGO) / duration(),    //EN Public Transfers Outflows (TGO)
        weighted_duration(current_TFB) / duration(),    //EN Net Interhousehold Transfers (TFB)
        weighted_duration(current_TFW) / duration(),    //EN Net Intrahousehold Transfers (TFW)
        weighted_duration(current_SF) / duration(),     //EN Private Saving (SF)
        weighted_duration(current_SG) / duration(),     //EN Public Saving (SG)
        weighted_duration(current_YL) / duration(),     //EN Labor Income (LY)
        weighted_duration(current_YAF) / duration(),    //EN Private Asset Income (YAF)
        weighted_duration(current_YAG) / duration()     //EN Public Asset Income (YAG)
    }
    * educ_fate +
};


table Person tabNtaVisualisationOld       //EN NTA Visualisation Old
[ in_nta_visualisation_year && nta_pop_group == NPG_OLD  && is_resident]
{
    nta_visual_year*
    sex+ *
    nta_family_old+ *
    {
        duration(), //EN Population
        weighted_duration(current_CFE) / duration(),    //EN Private Consumption Education (CFE)
        weighted_duration(current_CFH) / duration(),    //EN Private Consumption Health (CFH)
        weighted_duration(current_CFX) / duration(),    //EN Private Consumption other than
Education and Health (CFX)
        weighted_duration(current_CGE) / duration(),    //EN Public Consumption Education (CGE)
        weighted_duration(current_CGH) / duration(),    //EN Public Consumption Health (CGH)
        weighted_duration(current_CGX) / duration(),    //EN Public Consumption other than
Education and Health (CGX)
        weighted_duration(current_TGSOAI) / duration(), //EN Public Transfers Pensions, Inflows
(TGSOAI)
        weighted_duration(current_TGXCI) / duration(),  //EN Public Transfers Other Cash Inflows
(TGXCI)
        weighted_duration(current_TGXII) / duration(),  //EN Public Transfers Other In-Kind
Inflows (TGXII)
        weighted_duration(current_TGEI) / duration(),   //EN Public Transfers Education Inflows
(TGEI)
        weighted_duration(current_TGHI) / duration(),   //EN Public Transfers Health Inflows
(TGHI)
        weighted_duration(current_TGO) / duration(),    //EN Public Transfers Outflows (TGO)
        weighted_duration(current_TFB) / duration(),    //EN Net Interhousehold Transfers (TFB)
        weighted_duration(current_TFW) / duration(),    //EN Net Intrahousehold Transfers (TFW)
        weighted_duration(current_SF) / duration(),     //EN Private Saving (SF)
        weighted_duration(current_SG) / duration(),     //EN Public Saving (SG)
```

```
        weighted_duration(current_YL) / duration(),     //EN Labor Income (LY)
        weighted_duration(current_YAF) / duration(),    //EN Private Asset Income (YAF)
        weighted_duration(current_YAG) / duration()     //EN Public Asset Income (YAG)
    }
    * educ_fate +
};
```

The TablesPopulation.mpp Module

This module implements table output related to the total population and its composition

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
// Table Groups
////////////////////////////////////////////////////////////////////////////////////////////////
//

table_group TG_POPULATION_TABLES            //EN Population
{
    tabTotalPopulationAgeSexEduc
};

////////////////////////////////////////////////////////////////////////////////////////////////
//
// Tables
////////////////////////////////////////////////////////////////////////////////////////////////
//

table Person tabTotalPopulationAgeSexEduc   //EN Population by age, sex and education
[is_resident && is_alive && in_projected_time]
{
    sex + *
    educ3_level+ *
    {
        duration()                          //EN Population
    }
    * integer_age
    * sim_year
};
```