EUROMOD Basic Concepts

What is EUROMOD?

EUROMOD is a tax-benefit microsimulation model for the European Union (EU) that enables researchers and policy analysts to calculate, in a comparable manner, the effects of taxes and benefits on household incomes and work incentives for the population of each country and for the EU as a whole. As well as calculating the effects of actual policies it is also used to evaluate the effects of tax-benefit policy reforms and other changes on poverty, inequality, incentives and government budgets.

What can EUROMOD do?

EUROMOD can be used in many different ways in different contexts. Examples include:

Standard

- Estimation of poverty, inequality and redistribution statistics under actual conditions, previous or future tax-benefit rules
- Budgetary effects
- Effects of simple tax-benefit policy reforms (or illustrative changes to household composition and original income)
- "Model family" calculations
- Indicators of work incentives

More advanced

- Complex policy reforms (e.g. effects of revenue-neutral changes to tax rates and social insurance regulations)
- Policy swapping between countries (i.e. effects on country A of adopting a policy measure currently effective in country B) Generation of budget sets as input to labour supply or other models of behaviour change Generation of counterfactual income to answer "what if" questions Assessing effects and costs of EU-wide policy reforms.

Data imputation

- Generation of gross income (by source) from net, or vice versa

For further information see http://www.iser.essex.ac.uk/euromod.

EUROMOD input and output

EUROMOD output is essentially based on two inputs:

a. household micro-data and
b. EUROMOD *parameters*, which store rules on how to calculate taxes and benefits

Using these two information sources the model calculates all taxes and benefits that lend themselves for simulation. For example the income tax is calculated by taking the income the tax is based on from the dataset and applying the tax rules stored in the EUROMOD *parameters*, e.g. the tax schedule, on this base. These calculations are carried out for each household in the dataset and the result is written to a micro-output file. The output is at the individual level. It should however be taken into account that some of the results, for example family benefits, only make sense on household or family level. In such cases the benefit is usually assigned to the head of the assessment unit.

The main output of the model - disposable income for each household in the dataset - is made up of elements taken from the survey data - basically earnings and original income from other sources, as well as taxes and benefits that are not simulated - combined with taxes and benefits that are simulated by the model. The most common reason for not simulating a tax or benefit is lack of information, e.g. most pensions cannot be calculated by the model as the datasets usually do not contain information on past contributions to the pension system.

For each country implemented in EUROMOD there are:

a. one or more dataset(s), i.e. survey data collected in different years and
b. one or more *system(s)*, i.e. tax-benefit rules for different policy years.

Ideally, to calculate taxes and benefits for the year 2010 one would use the 2010 policy rules together with data referring to the year 2010. However, corresponding data is not always available and even if so, preparing and integrating new data in the model is a very laborious task. Therefore datasets are used for simulating several policy years, by up-dating monetary values to the corresponding policy year. Usually, for each *system* there is a dataset that is most suitable, normally the one whose collection year is nearest to the policy year. These optimal *system*-dataset combinations are referred to as *best match*. The EUROMOD *run dialog* shows which data and *systems* can be combined and which of these combinations are optimal. For further information on the EUROMOD *run dialog* and how to produce output with EUROMOD see Working with EUROMOD - Running EUROMOD.

EUROMOD stores its output in text files. As a default a standard output file is produced, which is called *cc_YYYY_std.txt*. *cc* stands for the country short name and *YYYY* stands for the tax-benefit year, e.g. *hu_2010_std.txt* if the output is based on the 2010 rules of the Hungarian tax-benefit system.

The output contains:

- identification numbers, household identifier (idhh), person identifier (idperson), identifiers of parents (idmother, idfather) and partner (idpartner)
- sample weight (dwt), which is the same for all household members
- demographic information, e.g. age (dag), gender (dgn), level of education (dec), economic status (les), etc.
- monthly original income (ils_origy)
- monthly disposable income (ils_dispy)
- other important aggregates on monthly basis, e.g. taxes (ils_tax), benefits (ils_ben), employee, employer and self-employed social insurance contributions (ils_sicee, ils_sicer, ils_sicse), earnings (ils_earns), public pensions (ils_pen), etc.
- several other informative variables (simulated or taken from data, demographic or monetary) and monetary aggregates
- possibly information on special assessment units, e.g. the inner family, (HeadID, IsDependentChild, IsParnter, etc.)

EUROMOD ***Country Reports*** document the way in which each country's tax-benefit system is modelled, i.e. the EUROMOD *parameters*. For further information see http://www.iser.essex.ac.uk/euromod/resources-for-euromod-users/country-reports.

Moreover, some simple statistics showing the redistributive effects of taxes and benefits in the EU, calculated by EUROMOD, can be viewed here:

http://www.iser.essex.ac.uk/euromod/statistics.

The EUROMOD user interface

The EUROMOD user interface is the central accession point of

the model. Its purpose is to help users to orientate themselves within the options provided by the model and around the model.

The user interface's main window shows the following

components:

- **The ribbon *Countries*** is filled with flags for the implemented countries. The flags can be clicked to access the EUROMOD parameters of the respective country. EUROMOD parameters store the information the model needs for its calculations. How the user interface presents this information is explained in Presentation of countries'

  tax-benefit-systems.
- **The button *Run EUROMOD*** left of the ribbon *Countries* opens the EUROMOD *run dialog*, which is explained in Working with EUROMOD - Running EUROMOD.
- **The ribbon *Country Tools*** enables access to a number of dialogs, which support working with a country's parameters. These dialogs allow for
  - configuring country settings (e.g. name, short

    name), system settings (e.g. currency, exchange rate to Euro) and the input datasets which can be used for simulating the country's tax benefit system (see section Working with EUROMOD - Changing Countries' Settings),
  - offer a number of search tools, which - besides

    allowing for standard searching and replacing - facilitate finding errors and assessing if and where certain EUROMOD components (e.g. variables) are used in the respective country's parameters (see section Working with EUROMOD - Searching),
  - offer a number of

    formatting tools, e.g. highlighting with colours and setting bookmarks (see section Working with EUROMOD -

Formatting).

- **The ribbon *Administration Tools*** enables access to dialogs which allow for:
    - adding and deleting of countries (see Working with EUROMOD - Adding countries and Working with EUROMOD - Deleting countries),
    - administrating EUROMOD's variables (see Working with EUROMOD - Administration of EUROMOD variables),
    - generating EUROMOD public versions (see Working with EUROMOD - Generating a EUROMOD public version).

- **The ribbon *Add-Ons*** is filled with icons for the available EUROMOD add-ons. The icons can be clicked to access the implementation of the respective add-on. The presentation of add-on implementations is equal to the presentation of country implementations and can therefore be looked up in Presentation of countries' tax-benefit-systems. For more information on add-ons and an example see EUROMOD Functions - EUROMOD add-ons and the special functions AddOn_Applic, AddOn_Pol, AddOn_Func and AddOnPar.
- **The ribbon *Applications*** enables access to a number of tools, which allow for analysing the EUROMOD output, respectively preparing the EUROMOD input (see section Working with EUROMOD - Applications). They comprise
    - a tool for generating simple summary statistics,
    - a tool for generating budget constraints,

    - a tool for generating hypothetical input data.

- **The main menu** left of the *Run EUROMOD* button contains functionalities like saving countries' parameters, opening projects, changing configuration, getting help and information as well as exiting the programme. These functionalities are described at appropriate points in the section Working with EUROMOD.

Presentation of countries' tax-benefit-systems

Loading countries

Clicking a flag in the user interface's *Countries* ribbon opens EUROMOD's representation of the respective country's tax-benefit system. The country’s name and flag is shown right of the *Run EUROMOD* button, to indicate which

country is loaded. Further countries can be loaded by clicking their flags.

Each country is represented in its own window and one can switch between these windows by again clicking the respective flag or via the Windows taskbar.

The EUROMOD spine

The main part of the window displays the representation of

the country's tax-benefit system - at the time of opening in a collapsed state.

Thus it shows what in EUROMOD lingo is frequently referred to as the *spine*. The EUROMOD spine is the list of *policies* worked off in sequence when the model performs its calculations. The term *policies* refers as well to policies in a narrow sense, i.e. benefits and taxes, as to special EUROMOD policies. These special policies for example define which variables are contained in the output. See Special EUROMOD policies below for further information. Each row of the spine represents one policy.

EUROMOD policies, functions and parameters

Expanding a policy by clicking the little *plus* button left of the policy's name allows viewing the implementation of the policy. A policy is implemented by so-called *functions*. Each EUROMOD

function is a self-contained building block that has its own parameters and represents a particular component of the respective policy. A typical social assistance benefit may for example be implemented by a function that determines eligibility for the benefit and a function that calculates the benefit amount for all eligible units. In fact these two functions: a function that determines eligibility/liability for benefits/taxes in a very general way and a function that calculates a wide range of benefits/taxes are the two most frequently used functions in EUROMOD. The purpose of using functions as building blocks of the model is to provide a general structure, which can be seen as using a standardised language to describe policy instruments. Once EUROMOD users are accustomed to this language, their understanding of other (foreign) countries'

benefits and taxes, and how they are implemented in EUROMOD, improves considerably. The parameters of a function can be viewed by expanding it using the little *plus* button left of its

name. The section EUROMOD Functions gives a detailed description of EUROMOD functions, including their use, the specific behaviour and the parameters they provide to guide this behaviour. Expanded policies and functions can be re-collapsed by the little *minus* buttons. Note that the policy and function worked on is displayed in the status bar.

EUROMOD systems

The changes of a country's tax-benefit system over time are

captured by EUROMOD *systems*. A

EUROMOD system either reflects the tax-benefit rules for a specific policy year (assuming annual adaptations of tax-benefit rules) or the rules for an actual (planned) or imaginary reform scenario. The columns of the country's view represent these EUROMOD systems. Thus the value of a specific parameter (e.g.

the rate of a particular tax band) for specific systems is defined at the intersection points of the parameter's row and the respective systems' columns (e.g. may take the value of 10% for the 2009-system and the value of 11% for the 2010-system).

The fact that policy reforms are not always as simple as

changing the value of a specific parameter is taken into account by so called *switches*. The intersection points of policies and functions with the systems represent these *switches*, i.e. they allow to switch policies and functions on or off dependent on the system. The *switches* also provide the settings *n/a*,

standing for not applicable, and *toggle*,

standing for temporarily switched off. More detailed explanations of these settings can be found in the section EUROMOD

Functions - Common Parameters.

Special EUROMOD policies

As mentioned above, the EUROMOD spine comprises policies,

which do not describe e.g. a tax or benefit. Instead they implement definitions, which are necessary for the model's calculation.

- The policy ***ILDef_cc*** contains

  definitions of so called *incomelists*.

  Technically an incomelist is the aggregate of several variables, which are added or subtracted to build the aggregate. The term "income"list indicates that the most common applications of this (important EUROMOD) concept are income definitions, for example disposable income, taxable income, etc.
- The policy ***TUDef_cc*** contains

  definitions of assessment units, which are in EUROMOD sometimes referred to as *tax units* (and are another important EUROMOD concept). Many taxes and especially benefits do not concern single individuals, but refer to bigger units, for example some sort of family definition. Assessment units allow for such definitions by specifying who belongs to the unit, who is a child, etc.
- Datasets are usually used for

  implementing several systems, by up-dating monetary values to the corresponding year. The policy ***Uprate_cc*** contains such up-dating of monetary dataset variables.
- The policies ***output_std_cc*** and ***output_std_hh_cc*** contain the specification of standard output on individual and household level (see section EUROMOD Basic Concepts - EUROMOD input and output).

The listed policies are standard, that means they are

implemented for each country contained in EUROMOD. Some countries contain further special policies like for example *ConstDef_cc*,

which defines basic values, which are used all over the implementation, e.g.

minimum income amount, pension age, etc.

Technically there is no difference between special policies

and standard policies, both use functions for their implementation and both need to be listed in the spine to be performed. Thus calling them special is just a matter of better comprehensibility. Moreover, the functions, which are in principle dedicated for the implementation of a special policy, like the function *DefIL* for the policy *ILDef_cc*, can be used in any other policy as well, if this seems appropriate.

EUROMOD terminology

The following index provides a brief description of some important EUROMOD specific terms and concepts.

EUROMOD system

The term system refers to the rules necessary to simulate a particular tax-benefit system. It may refer to an existing tax-benefit system (e.g. the UK tax-benefit rules for 2010) or to a reform scenario.

EUROMOD parameters

EUROMOD parameters contain the information the model needs to produce its output. Essentially they describe the tax-benefit systems of the implemented countries. They are stored in XML files (two for each country). Moreover, there are some common XML files, storing for example information about EUROMOD variables.

EUROMOD spine

EUROMOD spine is a term describing the list of taxes and benefits calculated by EUROMOD and the order in which they are processed.

EUROMOD assessment unit (tax unit)

The implementation of countries' tax-benefit systems usually requires assessment units. The smallest EUROMOD assessment unit comprises a single individual while the largest comprises the whole household. Some policy instruments, e.g. child benefits, require something in between, e.g. a family definition. EUROMOD assessment units also contain definitions of e.g. who is a dependent child, who is the head of the unit, etc. In EUROMOD lingo assessment units are frequently (and sloppily) referred to as tax units.

EUROMOD incomelist

EUROMOD incomelists are definitions of income concepts (e.g. disposable income) used within the tax-benefit system of a particular country. Technically an incomelist is the aggregate of several EUROMOD variables and possibly other incomelists. These components (in rare cases fractions or multiples of them) are either added or subtracted to build the aggregate. The term "income"list indicates that the most common applications of the concept are income concepts, for example disposable income, taxable income, …

EUROMOD policy

EUROMOD polices describe the implementation of particular taxes (contributions) or benefits of a country. Usually there is one policy for each tax or benefit. Polices are composed of EUROMOD functions. Apart from polices describing taxes and benefits, there are special polices, which define for example assessment units, incomelists or the content of output files.

EUROMOD function

EUROMOD policies are broken up into EUROMOD functions, which represent a particular component of the policy. As a (typical) example, a benefit policy may consist of a function that determines eligibility for the benefit and a function that calculates the benefit amount for all eligible units. The purpose of using functions as building blocks of the model is to provide a general structure, which can be seen as using a standardised language to describe policy instruments.

Standard output

As a default EUROMOD produces an output text files for each simulated system. This standard output file contains one row for each person listed in the input data, comprising some identification and demographic variables taken from the input data, as well as variables and incomelists calculated by the model, most essentially EUROMOD standard disposable income.

EUROMOD standard disposable income

In general the following components make up disposable income in EUROMOD (for each country and system): original income (essentially employment and self-employment income; capital, property and investment income; private pensions and transfers) plus benefits (cash transfers, essentially unemployment benefits, public pensions, family benefits, social transfers, other (country specific) cash transfers) minus direct taxes (essentially income tax, capital tax, other (country specific) direct taxes) minus social insurance contributions. As this income concept is standardised as far as possible over the countries implemented in the model it is referred to as standard disposable income (and defined in the incomelist ils_dispy).

EUROMOD variables

EUROMOD knows four types of variables: 1) variables contained in input data, 2) variables simulated by the model (marked with the postfix _s), 3) intermediate variables and 4) special purpose internal variables. Variables of type 1 and 2 are described in a special EUROMOD parameter file, the variable description file. Variables of type 3 are defined by using special EUROMOD functions (DefConst, DefVar). Variables of type 4 are produces internally by the model to fulfil specific functionalities, an example is the loop counter of the EUROMOD looping functions (Loop, UnitLoop).

Best match

The term best match describes an optimal system-dataset combination, as there is sometimes more than one possibility to simulate a system. For example, if the Belgian 2006 system can be simulated either by using data with income year 2005 or data with income year 2006, the combination 2006 system / 2006 dataset constitutes the best match. EUROMOD good practise however suggest setting the best match flag only for baselines, which are explained in the next paragraph.

Baseline

Baseline is usually the term used for a system-dataset combination, which fulfils the best match criterion as described above. In addition however, the system must refer to an actual policy year and the system-dataset combination must be the main or default implementation for the respective policy year. To understand this, assume a country for which three implementations for the policy year 2013 exist: (A) is using SILC data with income year 2013, (B) is using a national data source, also with income year 2013, while (C) is using the same data as (A), i.e. SILC data, but implements a reform scenario. Though each of the three systems is in line with the description of "best match", i.e. all use data with 2013 income, only (A) is called a baseline: (B) does not fulfil baseline criteria as it is not using the standard EUROMOD data source and (C) is not referring to an actual policy year. It must however be mentioned, that the term baseline is not a very clear definition, thus it could for example be used for (B) if a reform scenario exists, which also uses the national data source, in order to denote that (B) is the base scenario for the reform.

What concerns good practise in labelling a system-dataset combination as best match, only (A) should be labelled as such. Doing so allows for distinguishing between the EUROMOD core implementation and other developments. In the example, (B) does not belong to the core as it is a special development for a certain country, where some alternative data source can be used (for whatever reason). (C) is a reform scenario and therefore obviously not part of the core.

Working with EUROMOD

Running EUROMOD

To run EUROMOD open the run dialog by clicking the *Run EUROMOD* button in the left top corner of the user interface.[1]

Viewing and selecting systems to run

The main part of the dialog is formed by a list of systems which are ready to run. The content of this list depends on where the dialog was opened. If the *Run EUROMOD* button is pressed while no country is loaded, all systems of all countries are displayed. Whereas, if the button is pressed in a specific country, the systems of this country are displayed. This selection can however be changed via the ribbon bar, showing all countries. The country/ies whose systems are currently listed are marked by a light blue background. To select a country, i.e. to list the country's systems, just click it. Vice versa click a selected country to unselect it, i.e. to not any longer list its systems.[2]

To select a system for running check the tick-box right of the system. Note that some systems may be marked with red colour. These are systems that are set as private. Before however, you may want to determine the dataset which is applied for running the system. For this purpose the system list provides a combo-box for each system, which contains all available datasets. As a default these boxes show the best matching datasets (see EUROMOD Basic Concepts - Terminology). It is however possible to choose another dataset by selecting it from the list.[3]

The purpose of the button groups left of the *Run* button is to facilitate the selection of many countries and/or systems. The two top-most buttons allow for the selection of all respectively no country. The two buttons in the middle select respectively unselect all displayed (ordinary) systems for running. Finally the two buttons on the bottom select respectively unselect all displayed add-on systems for running.

Selecting the output path

The field *Output path* at the bottom of the dialog defines the folder where the model writes its output to. As a default the field shows the output folder defined via the main menu's item *Open project* (see Working with EUROMOD - Open project). The folder can be changed by clicking the folder button right of the field and selecting the folder via a dialog, or by typing. Please note that the output folder must exist, otherwise EUROMOD issues an error message.

Running the selected system-dataset combinations

Once the respective system-dataset combinations are selected, clicking the button with the green arrow in the ribbon bar starts EUROMOD's calculations. An info-window appears, which informs about the progress and allows for some manipulation. The window lists all system-dataset combinations selected for running and shows their status: *running*, *queued*, *finished* or *aborted* (either by the user or due to an error). Once a run is started, its starting time is displayed, and once it is finished (or aborted), the finishing time is indicated as well, together with the time taken.

Moreover, there are three buttons for each run. The *Stop* button allows for aborting the run. Once a run is started, the *Run Log* button is activated. If it is clicked, the field below the list of runs shows progress information. If a run produces an error (stopping the run) or a warning (allowing to continue the run) the *Error Log* button is activated. Clicking the button shows the run's warnings and/or errors in the field below the list of runs. Note that the content of this field is determined by the most recently clicked button - its heading indicates what is currently displayed.

Note that the info-window will stay open until it is closed by the user, even if all runs are finalised, to allow checking possible error logs, respectively inform about the times taken. If the user closes the window before all runs are finished, (after a respective warning) the still active runs are aborted and the queued runs are taken from the queue. To hide the window, use the minimise button.

Also note that, apart from the output files, the model produces a header file, which contains one row of information for each output file. The information includes: System (e.g. es_2011); Database (es_2007_a3.txt); EUROMOD-Version (e.g. f4.22); User-Interface-Version (e.g. 1.6); Executable-Version (e.g. 1.6);[4] Start (e.g. 11 Oct 2011; 12:10:58); End (e.g. 11 Oct 2011; 12:14:00); Outputfile (e.g. c:\euromod\euromodfiles\output\es_2011_std.txt); Currency (e.g. euro); Exchangerate (e.g. 1.00). The name of this file is yyyymmddhhmm_EMHeader.txt (e.g. 201110111210_EMHeader.txt). Moreover, if any warnings or errors were issued, an error-log file is generated, named yyyymmddhhmm_emerrlog.txt (e.g. 201110111210_emerrlog.txt).

Limiting the displayed datasets and systems

The ribbon *View / Filter / Add-Ons* provides options to limit the datasets and systems displayed.

If the field *Filter Datasets* contains a selection criterion the datasets offered for selection are limited to those whose name matches the criterion. The selection criterion may use * (for any letter) and *?* (for one arbitrary letter). For example, with the selection criterion set to **2009** only 2009 datasets, with the selection criterion set to **200?* only datasets from 2000 to 2009 are listed.[5]

If the field *Filter Systems* contains a selection criterion the systems displayed are limited to those whose name matches the criterion. As for datasets the selection criterion may use * (for any letter) and *?* (for one arbitrary letter).

If the checkbox *Best Match Only* is activated, only systems with a best matching dataset are displayed. Moreover, the combo-box with datasets lists (the usually unique) best matching dataset(s) only. For information on the "best match" criterion see EUROMOD Basic Concepts - EUROMOD terminology.

If the checkbox *Regular Expression* is activated, then the two fields *Filter Datasets* and *Filter Systems* will be treated as regular expressions. Regular Expressions are more complex to define, but allow for much more advanced filters. You can find a full description of how Regular Expressions work here: https://en.wikipedia.org/wiki/Regular_expression#Basic_concepts.

The fact that a(ny) filter or *Best Match Only* is set is indicated by a respective image appearing and by red colour.

Limiting the output to selected households

If the box *Show selected HH options* in the ribbon *View / Filter / Add-Ons* is checked, two additional columns are displayed in the system list: *First HH-ID* and *Last HH-ID*. They allow for limiting the output from the whole set of households contained in the input data to the households with household IDs (*idhh*) equal to or larger than *First HH-ID* and equal to or smaller than *Last HH-ID*.[6]

Running add-on systems

The section *View Add-Ons* in the ribbon *View / Filter / Add-Ons* provides a list with available EUROMOD add-ons. If one or more add-ons are selected, additional run-checkboxes are displayed in the system list (one for each add-on[7]). Depending on whether the add-on is available for the system, these boxes are enabled for selection or not. Note that add-on-systems can be run independent of standard tax-benefit calculations. For more information concerning add-ons see EUROMOD Functions - EUROMOD add-ons and the special functions AddOn_Applic, AddOn_Pol, AddOn_Func and AddOnPar.[8]

Advanced settings

The ribbon *Advanced Settings* allows for configuring some advanced options with respect to the model run:[9]

Do not stop on non-critical errors: Checking this box effects that the model does not stop on non-critical errors. Note that it still stops on critical errors. For more detailed information see Working with EUROMOD - Finding errors.

Add date to output-filename: Checking this box effects that yyyymmddhhmm is added to the name of the standard output file. For example, if the run of the Spanish 2011 system starts on 11 October 2011 at 12:10, the standard output file is called es_2011_std_201110111210.txt. This option allows to avoid overwriting the output with each model run. Moreover, the header file produced with each model run has the same ending (yyyymmddhhmm), thus a direct link between header file and corresponding output files is generated.

Log runtime in detail: Checking this box advises EUROMOD to produce a detailed run-time-log. That means, instead of only logging total runtime, the model also logs runtime for single policies and functions and writes this information into the header file. Please note that the durations measured by this logging procedure are somewhat distorted as time recording itself takes time (though not very much). As a consequence it is recommended to switch time recording off for time critical runs.

Close dialog after run: Checking this box effects that the *Run EUROMOD* dialog is closed after launching the programme.

Do not pool system's datasets: Checking this box forces each system-dataset combination to be shown in a separate row.[10] This (longer and less concise) listing has the advantage that a single system can be run with different datasets more conveniently. To avoid, that the output of a systemX-datasetA combination is overwritten by the output of a systemX-datasetB combination (as the name of the default output file is the same), the output is arranged in folders named after the datasets. That means there would be one folder named after dataset A, containing systemX output produced with dataset A and one folder named after dataset B, containing systemX output produced with dataset B.

Run public components only: If this box is checked, any private policies, functions or parameters are ignored, i.e. the result is the same as if they did not exist. Please note that ticking the box does not hide any private systems or datasets (for performance reasons).

Parallel runs: This field indicates how many instances of the EUROMOD executable run in parallel. For example, if *parallel runs* is set to three and five systems (using different datasets) are selected for running, three runs are started immediately, while the remaining two runs are queued. By default this option is set to "Auto" in which case the application will automatically determine the most efficient number of parallel runs based on the number of available CPU cores.

Using policy switches

The section *View Policy Switches* in the ribbon *View / Filter / Add-Ons* provides a number of checkboxes, which allow for switching on or off certain parts of the EUROMOD calculations. Checking, for example, the box *Tax Compliance Adjustments* extends the list of systems by a column, which provides a button for each system. These "switch buttons" are captioned by either *on* or *off*. An *on* button indicates that correction for tax compliance is part of the system's calculations. If users want the model to omit these calculations (i.e. see what happens if everybody adheres the rules of tax legislation) they can do so by clicking the *on* button – this changes the caption of the button to *off*, indicating that the next model run will not carry out the parts of calculation, which correct for tax compliance. The same applies for the other checkboxes, thus allowing e.g. taking (not) into account benefit non-take-up, etc.

Moreover a switch button may be captioned by *on (default)* or *off (default)*. This indicates that the current setting is the default, i.e. if the switch button for tax evasion is captioned by *on (default)* for a system means that usually tax evasion is part of the system's calculations. The button *Restore Defaults* left of the checkboxes for viewing policy switches allows for setting all switches back to their defaults. Alternatively a user may use the context menu (mouse right-click) in any policy switch column to change the switch for all systems to on, off or the default value of each system. For more information on how this is accomplished technically see Working with EUROMOD - Changing Countries' Settings - Administrating policy switches.

Furthermore a button may also show no caption. This means that the policy is not switchable for one of two reasons: Firstly, the country (or system) may not foresee respective calculations. For example, correction for tax compliance may not be implemented for Denmark or it may (for whatever reason) just not be implemented for the Danish 2009 system. Secondly, though the policy is available and in principle switchable, it is not for this specific country (or system). For example tax evasion may be implemented for Greece, but it is a compulsory part of the tax-benefit calculations and thus not switchable via the run dialog.[11] Again, for more information on how this is accomplished technically see Working with EUROMOD - Changing Countries' Settings - Administrating policy switches.

As the possibility to display more than one such button column suggests, it is possible to combine different settings, e.g. taking (not) into account tax compliance as well as benefit take-up.

Note that the settings are not only system specific, but in fact system-dataset combination specific. System specificity implies that the caption of the buttons may be different amongst the systems of a country. Dataset specificity means that a system's button may change if another dataset is selected.

Also note that the user interface remembers users' settings of switch buttons. That means if a user for example changes a switch button from *on (default)* to *off* and then closes the run dialog or even the whole interface, next time she opens the run dialog and displays the respective column, the button will still be set to *off*.

Moreover note that the header file (see paragraph *Running the selected system-dataset combinations* above) contains a column *Policy Switches*, which reports the concrete setting of the policy switchesfor the run, by entries like e.g. *bta_??=on;tca_??=off*.[12]

Finally note that switch buttons do not only apply for "ordinary" systems, but also for add-on systems. That means one could for example calculate marginal tax rates with tax evasion switched on or off.

Finally, finally note that, if the switch buttons for a switchable policy are not visible (because the column is not displayed), the model always applies the default settings, irrespective of any changes by the user. That means, to avoid misleading results by forgotten policy-switch-changes, changes by the user are only effective if she sees in the run dialog that they are set.

[1] You may have to switch to the ribbon *Countries*, if the button is not visible.

[2] Note that this selection is preserved if the dialog is closed. If, you are for example working with Ireland and for some reason also select UK systems to be shown and then close the dialog, on the next opening (within the same EUROMOD session) UK as well as Irish systems will be displayed. In this context note that the dialog is country specific, i.e. the respective selection belongs to one country. In other words, if you then open Estonia, it still shows only the Estonian systems and not the UK ones.

[3] If no best matching dataset is defined for the system the first best dataset is displayed. Note that only systems with at least one dataset assigned are listed.

[4] User-Interface-Version and Executable-Version are in fact equal. The reason for having two numbers is historical - formerly user interface and executable were not delivered as an integrated software package. A replacement of the two equal numbers by a single "EM-Software-Version" may be considered for the future.

[5] Note that limiting datasets may also limit systems, as a system for which no dataset is available will not be displayed.

[6] Note that the selection of households is database specific, i.e. if several systems are run with the same dataset, the household limitation of the first system (if specified) is used, limitations of other systems are ignored.

[7] To be precise, there may be more than one checkbox for each add-on as add-ons may contain several systems. Thus there will be one checkbox for each add-on system.

[8] Technically running add-ons is accomplished as follows: The user interface reads all *AddOn_xxx* functions and follows their instructions with respect to merging the add-on system with the selected (base) system. The result of this merging is saved in the temporary folder. Consequentially the user interface instructs the executable to run the system from there.

[9] Note that these settings are stored with the local implementation of the user interface, i.e. they are preserved and will be the same if the interface is closed and opened again. Moreover, they are not country specific, i.e. if they are e.g. set for Bulgaria and afterwards the run dialog is opened for Hungary, the "Bulgarian" settings are overtaken for Hungary.

[10] If for example, the system *cc_yyyy* can be run with datasets *cc_yyy0_a1* as well as dataset *cc_yyy1_a1*, it would usually be listed once with the two datasets selectable via the combo-box (with the best match being preselected). If *Do not pool system's datasets* is checked, there would be two rows: one for the combination of system *cc_yyyy* with dataset *cc_yyy0_a1* and one for the combination of system *cc_yyyy* with dataset *cc_yyy1_a1*.

[11] In this case, as usual, the switch in the country file is relevant. That means the policy can be set "permanently" to on, off, toggle or n/a.

[12] More precisely it reports for all available policy switches, whether they are turned on or off. If, for example, for a system *Tax Compliance Adjustments* as well as *Benefit Take-up Adjustments* are switchable, the header file will report something like *bta_??=on;tca_??=off*, independently of the fact whether the user has changes the switches or defaults were applied. Whereas, if for a system only *Tax Compliance Adjustments* are switchable (because *Benefit Take-up Adjustments* do not exist or are not switchable), the header file will report something like *tca_??=off*.

Changing countries' tax-benefit-systems

The sub-sections of this section explain the different functionalities of the EUROMOD user interface, which support implementing and changing a country's tax-benefit systems in EUROMOD. The order of sections is arbitrary and in general not based on each other.

Adding Systems

Adding a system via copy/paste

To add a system either click the *Add System* button in the ribbon *Country Tools* or click the a(n existing) system's header to select the menu item *Copy/Paste System* from the context menu. This opens a dialog where you are asked to indicate the new system's name. Note that the system's name must not contain any other characters than letters, numbers and underscores. If any other character is used or if the chosen name is equal to an existing system's name, an error message is issued, and you are asked to change the name. Clicking *OK* adds the new system.

Note that new EUROMOD systems are initially always a copy of an existing system, as it is very likely that the new system can use an already implemented system as template. For example, a country's 2012 system can be based on the already implemented 2011 system, as well as a reform scenario is usually based on an existing system. Such a template system is usually referred to as the *base system*. If the new country is added via the context menu of an existing system, the respective system serves as the base system. If the new country is added via the *Add System* button one will be prompted a dialog allowing for the selection of the base system.

The new system is initially nearly a true copy of the existing system. That means amongst others that the new system is automatically configured to run with all datasets the base system is configured to run with. The only difference between the base systems and their copies are the names of standard output files. If the base system, for example, is called *sl_2010* it usually produces a standard output file called *sl_2010_std.txt*. For the derived system, called for example *sl_2010reform*, this name is changed to *sl_2010reform_std.txt*. For more information on standard output see EUROMOD Basic Concepts - EUROMOD input and output.

Note that the user interface offers the possibility to highlight the differences between a base and a derived system, using background and/or text colour. If the base system highlights differences to its own base system, it inherits this feature to the new system. Also other conditional formats are overtaken from the base system. For more detailed information see Working with EUROMOD - ConditionalFormatting.htm.

Adding the very first system

The context menu, which is opened on clicking the *Policy column* offers the menu item *Insert First System*. This menu item is disabled unless the country does not yet contain any system. Select the menu item to open a dialog where you are asked to indicate the new system's name and click *OK* to add the system. Note that the same naming conventions are valid as explained above.

Renaming a system

To rename a system click the system's header to select the menu item *Rename System* from the context menu. This opens a dialog where you are asked to indicate the system's new name. Note that the system's name must not contain any other characters than letters, numbers and underscores. If any other character is used or if the chosen name is equal to an existing system's name, an error message is issued, and you are asked to change the name. Clicking *OK* renames the system.

If the system contains standard output policies (for example the system *BG_2014* will usually contain the output policy *output_std_bg* with the output file named *BG_2014_std*) you are asked whether the standard output file(s) should be renamed. You can allow for this action or refuse.

Moreover, if you change the year of a systems (e.g. rename *BG_2014* to *BG_2015*) you are informed that *"... uprating-indices are now updated in accordance with your changes to system-year"*. You can refuse this action, but it is not recommended (see Working with EUROMOD - Defining uprating factors).

Finally, you are informed that *"… Add-Ons are not adapted to reflect the new system name. Please change policy 'AddOn_Applic' manually where necessary. "*. You can avoid this warning for the next time by ticking the option *Do not reshow this warning* (see Working with EUROMOD - Configuration).

Deleting systems

To delete systems click the *Delete System(s)* button in the ribbon *Country Tools*, which will prompt you a dialog allowing for selecting the systems. Alternatively right click the system's header to open the *system context menu* and select the menu item *Delete System*. Answering the security query with *Yes* deletes the selected system(s).

Cleaning systems

To clean-up your systems, select the ribbon *Country Tools* and click the button *Clean Up System*. This will automatically search for policies, functions and parameters that are set as *n/a* in all systems and it will remove any and all such instances. Note that as this is an action that cannot be undone, the user interface produces a backup before starting the action, which can be restored via the button *Restore* in the ribbon *Country Tools*. For more information see (Working with EUROMOD - Backup - Restore).

Changing the left to right order of systems

The order of systems from left to right is changed by using drag and drop, i.e. clicking a system's header with the left mouse button, dragging the system with pressed mouse button to the new position and dropping it there by releasing the mouse button. Note however, that this new position is not permanent. This means that, if the country is closed and opened again, the system will be back to its old position, even if the country is saved.

Saving a new order of systems

To make a new order of systems permanent, click any system's header to open the system context menu and select the menu item *Save System Order*. Note that this change is initially only valid for the current session (see *Restore System Order* below). To become valid for further sessions the country must be saved.

Restoring the initial order of systems

One can restore the order of systems by right clicking any system's header to open the system context menu and selecting the menu item *Restore System Order*. The thus restored order is the order one came upon when the country was initially opened, unless meanwhile a new order was stored via *Save System Order* (see above), in which case this order is restored.

The Hidden Systems Box

The system context menu's item *Move To Hidden Systems Box* ... shows a couple of sub menu items[1], which provide options for hiding and unhiding systems from the main view. These options are helpful for reasons of manageability, for example if a reform scenario is implemented and one wants to concentrate on this system and maybe its base system.

Hiding Systems

Systems hidden from the main view are listed in the *Hidden Systems Box*. This is a small window, which is displayed by clicking on the *Show Hidden Systems Box* button in the *Display* ribbon or by choosing the sub menu item *Show Hidden Systems Box* or by any of the following sub menu items. Note that the term "selected system" used

below refers to the system that was right clicked to open the system context menu.

- **Sub menu item *Selected System*:** Selecting this option hides the selected system.
- **Sub menu item *All Systems But Selected*:** Selecting this option hides all other systems, while only the selected systems remains visible.
- **Sub menu item *Select Systems* ...:** Selecting this option provides a list of all visible systems and allows selecting them for hiding. Note that clicking *OK* without selecting any system, though not having any effect on system visibility, may still (re)show the *Hidden System Box* (if it was closed via its close box).
- **Using Drag and Drop:** One can also hide systems by dragging them into the *Hidden System Box*, i.e. clicking the respective system's header with the left mouse button, dragging the system with pressed mouse button over the *Hidden*

  System Box and dropping it there by releasing the mouse button.

Unhiding Systems

- **Sub menu item *Unhide all Systems*:** Selecting this options reshows all hidden systems, i.e. the systems listed in the *Hidden System Box*.
- **Using Drag and Drop:** One can also unhide a

  system by dragging it from the *Hidden System Box* to its old or any other position (see Working with EUROMOD -

  Changing the order of systems for more information with respect to the order of systems).
- **Double clicking the system in the *Hidden System Box*:** This reshows the system and shows it as the last one, i.e. just left of the comments column.

Note that the user interface draws the attention to any

changes, which could affect hidden systems. If for example a function is deleted a warning will be issued, which asks you to *"Please note that the action will have effect on the hidden systems as well!"*. (See Working with EUROMOD - Managing warnings for avoiding such warnings.)

[1] These sub menu items are accessed by firstly right clicking a(ny) system's header to open the system context menu and then moving the mouse over the menu item *Move To Hidden Systems Box* ....

Matrix view of incomelists

The "matrix view" of a system's incomelists shows a tabulate view of this system's incomelists, where the headers of the columns contain the names of the incomelists and the headers of the rows contain the names of all variables included in any incomelist. The cells at the crossing points contain the "contribution" of the respective variable to the respective incomelist, e.g. 1, -1, 0.5, -0.5.

The view is opened by right clicking a system, to open its context menu, and selecting the menu item *Show Matrix View of Incomelists*. Another option to open the view is by clicking the button *Show Matrix View of Incomelists* in the ribbon *Display* and selecting the respective system from the appearing list.

Note that the window showing the matrix view of incomelists is non-modal, that means the window can stay open while editing. Use respective resizing, the *minimise*, *maximise* and *close* button to position the window where it is convenient.

Selecting components and values

There are a number of operations which can be applied on several components (policies/functions/parameters) or values. For example one can delete a range of parameters in one go or copy the values of several parameters.

For this purpose it is necessary to select the respective components and/or values, respectively the corresponding range of cells in the display. This is accomplished by selecting the first cell of the range, and then pressing and holding the *Shift* key while selecting the last cell of the range. Light blue back colour will show the selection.

The selected region can be 'manually' unselected by either pressing the Escape-key or by selecting a cell outside the region. The selection is also automatically cancelled by several other actions, in particular actions, which cause (or risk causing) a 'broken' selection, i.e. the selected cells are not placed in a single rectangle anymore (also see below).

The following notes and hints may be useful in understanding how selecting works:

- The 'first' and 'last' cell of the selection (which are to be selected by the user) build the top-left and bottom-right end points of the selection, and the selection will contain all cells in between them. It is however not necessary to select the top-left end point first. In other words selecting from the cell in row A, column B to the cell in row X, column Y leads to the same result as selecting from the cell in row X, column Y to the cell in row A, column B.
- The selection (and the related operation) does only concern visible cells. Cells between the end points, which are hidden due to collapsing or hidden columns, are not part of the selection. Therefore, to avoid a 'broken' selection, any selection is cancelled by collapsing or expanding.
- Instead of using the mouse for selecting the first and last cell of the selection the arrow keys (left, right, up, down) and the position keys (page down, page up, etc.) can be used. The selection starts when such a key is pressed together with the *Shift* key and ends once the *Shift* key is released.

Adding Policies

Adding a policy

To add a policy right click the name of the policy before

respectively after which you want to insert the new policy. This opens the policy's context menu, where you select the menu item *Add Policy Before* respectively *Add Policy After*. A sub menu opens, which allows choosing the type of the new policy (benefit, tax, ...). Click the respective policy type to open a dialog where you are asked to indicate the new policy's name. Clicking *OK* adds the new policy. Note that the policy's name must not contain any other characters than letters, numbers and underscores. If any other

character is used or if the chosen name is equal to an existing policy's name, an error message is issued, and you are asked to change the name. Moreover, the policy name is by convention expected to end with an underscore followed by the country's short name (e.g. _hu, _be, _uk, etc.). If this is not the case the user interface asks whether it should add this ending for you. You may answer this question with *No* to use a

non-standard policy name, it is however recommended to answer with *Yes*.

Adding the very first policy

The system context menu (opened by right clicking any system's

header) offers the menu item *Insert First*

Policy. This menu item is disabled unless the country does not yet contain any policy. Select the menu item to open a dialog where you are asked to

indicate the new policy's name and click *OK* to add the policy. Note that the same naming conventions are valid as explained above.

Renaming a policy

To change the name of a policy right click its current name.

This opens the policy's context menu where you select the menu item *Rename Policy*. A dialog opens where you are asked to indicate the new name of the policy. Click *OK* to confirm the change. Note that the same naming conventions are valid as explained above.

Deleting policies

Policies can be deleted by either using the context menu or via the *Delete*-key. In both cases first select the policy/ies you want to delete. Then press the *Delete*-key. Alternatively right click the name of (one of) the policy/ies for opening the context menu and selecting the menu item *Delete Policy/ies*.

A single policy is selected by simply clicking it (more precisely, its name in the *Policy* column). A range of policies is selected by selecting the first policy (i.e. its name in the *Policy* column), then pressing and holding the *Shift* key while selecting the last policy. Light blue back colour shows the selection. For a more detailed description of how to select cells see Working with EUROMOD – Selecting components and values.

Note that the selection needs to include the policy column (i.e. the name(s) of the policy/ies). If only system columns and/or the group or comment columns are selected, the interface assumes that the respective *values* should be deleted – see Working with EUROMOD – Changing parameters (paragraph *Deleting (ranges of) parameter values*). Moreover, if you use the context menu, you need to open it via a policy contained in the selection, otherwise not the selected policies are deleted, but the policy that belongs to the context menu.

A security question is issued before the policies are finally removed, which hints especially to the attempt of deleting compulsory policies.

Changing the order of policies

There are three ways to change the order of policies: by using (a) drag and drop, (b) key *Up* / key *Down* or (c) the policy context menu. For each of them first select the respective polices. A single policy is selected by simply clicking it (more precisely, its name in the *Policy* column). A range of policies is selected by selecting the first policy (i.e. its name in the *Policy* column), then pressing and holding the *Shift* key while selecting the last policy. Light blue back colour shows the selection. For a more detailed description of how to select cells see Working with EUROMOD – Selecting components and values.

For (a) click the respective policy/ies with the left mouse button, drag them with pressed mouse button to the new position and drop them there by releasing the mouse button.

For (b) press the *Control* as well as the *Up* key to move the respective policy/ies up, respectively the *Control* as well as the *Down* key to move them down.

For (c) right click the name of (one of the) policy/ies for opening the context menu and selecting the menu item *Move Policy/ies Up* respectively *Move Policy/ies Down*.

Note that changing the order of policies invokes a warning, asking "Are you sure you want to change the order of calculations?". See EUROMOD Basic Concepts - Presentation of countries' tax-benefit systems and EUROMOD Basic Concepts - Terminology for more information on the EUROMOD spine and the consequences of reordering policies. To avoid the warning see Working with EUROMOD - Managing warnings.

Copying policies

Copying policies within a country

To copy a policy right click its name for opening the policy context menu and selecting the menu item *Copy Policy*.

To paste the copied policy right click the name of the policy before or after which you want to insert the copy. From the policy context menu select the menu item *Paste Policy Before* to place the copied policy before the right clicked policy or *Paste Policy After* to paste the copied policy after the right clicked policy.

A dialog appears which asks you to indicate the policy's name, initially suggesting the template's name. Note however that you need to change this name as there must not be two policies with the same name. Moreover note the policy naming conventions explained in Working with EUROMOD - Adding policies. Clicking *OK* adds a new policy, which is initially (except the name) a perfect copy of its template.

Copying policies from one country to another

Copying a policy to another country is in principle not different to the approach described above, however after defining the name, a dialog appears which allows the assignment of systems. To understand this, consider a policy that is copied from a country with three systems to a country with just one system: which system’s parameter values should be copied?

The dialog contains a table, which lists the system assignments, where assignment means that the parameters of the destination system are to be filled with the values of the source system. The left part of each row shows a system of the destination country and the right part indicates the assigned system of the source country. If the destination country’s system is not assigned to any system of the source country, the right part of the table shows "Not Assigned". To choose another system of the source country, just click the respective cell, which opens a dialog that allows selecting the system. Select no system for no assignment.

Clicking the button *Clear Assignment* removes all assignments between destination and source systems, thus the table shows "Not Assigned" for all destination systems. Note that, when it is first opened, the dialog shows a default assignment, which tries to match systems of the same policy year, e.g. UK_2010 with BE_2010, UK_2012 with BE_2012, etc.

Clicking *OK* adds the policy as specified. If a system is not assigned, parameter values are set to *n/a*.

Copying policies as a reference

The policy context menu also offers the menu items *Paste Reference Before* respectively *Paste Reference After*. The case is as follows: it may be necessary, that a policy is calculated twice in the model run. For example, social insurance contributions may need to be calculated before income tax, as they are needed as an input, however income tax may as well be needed as an input for social insurance contribution calculations. A usual approach to solve such a problem of circularity is to calculate contributions twice, before and after income tax calculations. As the order of policies in the spine determines the processing sequence of policies (see EUROMOD Basic Concepts - Presentation of countries' tax-benefit systems and EUROMOD Basic Concepts - Terminology for more information on the EUROMOD spine and the order of policies) it would be necessary to insert the policy calculating social insurance contributions twice. Such a redundancy is however error prone as changes need to be implemented in both policies, which is easy to forget. A reference avoids this problem by being just an entry in the spine with the same name as the respective policy but with the twin symbol to denote it as a reference. The reference is just information for the rerun of a policy and does not display any functions - to view its definitions one has to refer to the "real" policy.

Technically inserting a reference works just like copying a policy, with the difference that the menu items *Paste Reference* are selected instead of *Paste Policy*. For obvious reasons it is not possible to copy a policy in one country and paste it to another country.

Setting policies, functions and parameters private

To set a policy, function or parameter private, respectively to remove this setting, right click its name for opening the context menu and selecting the menu item *Set/Unset Private*. If the policy/function/parameter currently is set private, the menu item shows a red symbol. In this case clicking the menu item removes the private setting and the symbol takes on its usual colour. Vice versa, if a policy/function/parameter is not private, the symbol shows its usual colour. In this case clicking the menu item installs the private setting and changes the symbol's colour to red.

For further information concerning private components see Working with EUROMOD - Generating a EUROMOD public version.

Changing policy view and jumping to a specific policy

The user interface allows for two modes for viewing a country's tax-benefit policies. The standard mode is "*Full Spine View*", that means all policies are displayed. For reasons of clarity or to enhance performance one may however prefer to view only one policy at once, i.e. work with "*Single Policy View*". The ribbon *Display* provides check-boxes allowing for changing between the two views.

If one changes from *Full Spine View* to *Single Policy View*, the single policy displayed is the focused policy. Initially the policy is fully expanded, that means all functions and parameters are displayed. To select another policy click the *Policy* column's header with the left mouse button. This opens a list of all policies, thus allowing for selecting the required one. The user interface "remembers" this policy during the session. That means if one switches back to *Full Spine View* and then again back to *Single Policy View* the displayed policy is the one displayed before the switches.

Working in *Single Policy View* is in principal not different from working in *Full Spine View*. Just a few operations are not possible and therefore deactivated. These operations include adding and deleting policies and operations which work on the full spine (e.g. importing/exporting systems).

Note that the list of all policies activated by a left mouse click on the *Policy* column is also available in *Full Spine View*, where it serves to jumping to the selected policy. This functionality may be handy if many policies and functions are expanded.

Adding a function

To add a function to a policy right-click the name of the function before respectively after which you want to insert the new function. This opens the function’s context menu, where you select the menu item *Add Function Before* respectively *Add Function After*. A sub menu allows choosing which function should be added.

This sub menu is slightly different depending on the policy the new function will be part of. For "normal" policies (e.g. benefits and taxes) firstly the policy functions are listed (*ArithOp*, *Elig*, *BenCalc*, etc.), followed by two sub menus containing *System Functions* (*DefConst*, *DefIL*, *DefOutput*, etc.) and *Special Functions* (*Loop*, *Store*, *Totals*, etc.). For special policies (e.g. *Output_std_cc* or *ILDef_cc*) the list starts with the function(s) associated with this policy (i.e. function *DefOutput* in the case of policy *Output_std_cc* and function *DefIL* in the case of policy *ILDef_cc*) followed by three sub menus containing *Other System Functions* (*DefConst*, *DefTU*, *Uprate*, etc.), *Policy Functions* (*ArithOp*, *Elig*, *BenCalc*, etc.) and *Special Functions* (*Loop*, *Store*, *Totals*, etc.). See EUROMOD Basic Concepts - Presentation of countries' tax-benefit systems for further information on special EUROMOD policies and EUROMOD Functions for further information on idem.

Click the respective function to add the function itself as well as the compulsory parameters of this function (e.g., for most functions, *TAX_UNIT* and *Ouput_Var*).

Alternative to using the function context menu the policy context menu offers the menu item *Add Function*. Using this adds the function as the very last function of the *policy*.

Deleting functions

Functions can be deleted by either using the context menu or via the *Delete*-key. In both cases first select the function(s) you want to delete. Then press the *Delete*-key. Alternatively right click the name of (one of) the function(s) for opening the context menu and selecting the menu item *Delete Function(s)*.

A single function is selected by simply clicking it (more precisely, its name in the *Policy* column). A range of functions is selected by selecting the first function (i.e. its name in the *Policy* column), then pressing and holding the *Shift* key while selecting the last function. Light blue back colour shows the selection. For a more detailed description of how to select cells see Working with EUROMOD – Selecting components and values.

Note that the selection needs to include the *Policy* column (i.e. the name(s) of the function(s)). If only system columns and/or the group or comment columns are selected, the interface assumes that the respective *values* should be deleted – see Working with EUROMOD – Changing parameters (paragraph *Deleting (ranges of) parameter values*). Moreover, if you use the context menu, you need to open it via a function contained in the selection, otherwise not the selected functions are deleted, but the function that belongs to the context menu. In addition note that with the context menu it is only possible to delete functions belonging to one policy – use the *Delete*-key if you want to delete a wider range of components.

A security question is issued before the functions are finally removed. Note that you can opt to not show the security query anymore - see Working with EUROMOD - Managing warnings.

Changing the order of functions

There are three ways to change the order of functions: by using (a) drag and drop, (b) key *Up* / key *Down* or (c) the function context menu. For each of them first select the respective functions. A single function is selected by simply clicking it (more precisely, its name in the *Policy* column). A range of functions is selected by selecting the first function (i.e. its name in the *Policy* column), then pressing and holding the *Shift* key while selecting the last function. Light blue back colour shows the selection. For a more detailed description of how to select cells see Working with EUROMOD – Selecting components and values.

For (a) click the respective functions with the left mouse button, drag them with pressed mouse button to the new position and drop them there by releasing the mouse button.

For (b) press the *Control* as well as the *Up* key to move the respective functions up, respectively the *Control* as well as the *Down* key to move them down.

For (c) right click the name of (one of the) functions for opening the context menu and selecting the menu item *Move Functions Up* respectively *Move Functions Down*.

Note that functions can only be shifted (up or down) within the policy they belong to. That means, this way it is not possible to move a function from one policy to another. It is however possible to copy a function into another policy (see Working with EUROMOD - Copying functions) and possibly (if the intention is to move the function) delete the original function.

Also note that changing the order of functions invokes a warning, asking "Are you sure you want to change the order of calculations?". See EUROMOD Basic Concepts - Presentation of countries' tax-benefit systems and EUROMOD Basic Concepts - Terminology for more information on the EUROMOD spine and the consequences of reordering functions. To avoid the warning see Working with EUROMOD - Managing warnings.

Copying functions within a country

To copy a single function right-click its name, which opens the function's context menu, where you select the menu item *Copy Function(s)*.

To copy several functions, first select the respective functions, then click within the selection to open the context menu providing the menu item *Copy Function(s)*. Note that the selection needs to include the policy column and that the right-click must be performed in the policy column as well. Moreover, note that only functions of one policy can be copied at the same time - a warning is issued if the selection does not fulfil this criterion. (For a description of how to select functions see Working with EUROMOD – Selecting components and values.)

To paste the copied function(s) right-click the name of the function before or after which you want to insert the copy. From the context menu select the menu item *Paste Function(s) Before* to place the copied function(s) before the right clicked function or *Paste Function(s) After* to paste the copied function(s) after the right clicked function.

Also the policy context menu (opened by a right-click on the name of a policy) provides the menu item *Paste Function(s)*. Using this option adds the copied function(s) at the end of the policy.

Copying functions from one country to another

Copying function(s) to another country is in principle not different to the approach described above, however on pasting the function(s), a dialog appears which allows the assignment of systems. For a description of the dialog and its handling see Working with EUROMOD - Copying policies, paragraph *Copying policies from one country to another*. Clicking *OK* adds the function(s) as specified.

Displaying function specifiers

The ribbon *Display* provides a button *Show Key Parameters*. Clicking this button shows an additional row below each functions' name, displaying the "key parameter" of the function. For example the row below a *DefIL* function shows the name of the respective incomelist. More precisely it shows for example *Name: ils_earns*. That means it indicates the name of the parameter hosting the key parameter (parameter *Name*), followed by the parameter's value (*ils_earns*). If the value of the parameter is not equal for all systems, all values are displayed, e.g. *Name: ils_earns*, *ils_earns_2008*, *ils_earns_2012*.[1]

For most of the functions the parameter *output_var* (respectively *output_add_var*) constitutes the key parameter. Exceptions are the functions *DefIL* and *DefTU* with parameter *Name* as key parameter, the functions *Uprate* and *SetDefault* with parameter *Dataset*, the function *DefOutput* with parameter *File*, the function *Elig* with parameter *Elig_Cond* as well as some other less frequently used functions. Some functions do not have a key parameter, as for example functions *DefVar* and *DefConst*. For these functions no additional row is displayed.

To hide the key parameters press the button *Hide Key Parameters* (which is the *Show Key Parameters* button, which changed its name while key parameters were displayed).

Note that this change of display affects all open countries, e.g. if key parameters are displayed for UK while Hungary is open too, key parameters are displayed for Hungary as well.

[1] The information also takes care of the fact that specification parameters may appear more than once, like e.g. the parameter *Dataset* for functions *Uprate* and *SetDefault*.

Presentation of (normal and) special parameters

Each

parameter is represented by one row in the spine. (See EUROMOD Basic Concepts - Presentation of countries' tax-benefit systems and EUROMOD

Basic Concepts - Terminology for more information on the EUROMOD spine and the order of policies and functions.)

- The **Policy column** of this row takes the name of the parameter, for example *Output_Var* for

  the parameter taking the name of the output variable of the respective function or *LowLim* for taking a lower limit

  for the result of the function. In general the *Policy* column is not editable. See Parameters with editable parameter name below for an exception.
- The **Grp/No column** is described in the paragraphs Group parameters and Footnote parameters below.
- The **Comment column** may contain a note or remark with respect to this specific parameter.
- The **System columns**,

  i.e. the columns between the *Grp/No* and the *Comment* column, contain the value of the parameter for each respective system.

  Parameter values range from simple numbers (e.g. the rate of a particular tax band) to complex formulas (e.g. a condition describing the eligibility for a certain benefit). For detailed information see EUROMOD Functions - Types of parameter values.

  For more information on EUROMOD systems see EUROMOD Basic Concepts - Presentation of countries' tax-benefit systems and EUROMOD

  Basic Concepts - Terminology.

Group parameters

Parameters of a

function may form a group in the sense that they can describe their information only in combination, or that parameters of the group provide additional information to the information described by other parameters of the group.

Examples for the former are the parameters *Comp_Cond* and *Comp_perElig* of the function *BenCalc*, in the sense that the condition *and* the amount form the information

that makes up the component. An example for the latter is the parameter *Comp_UpLim*, which is additional information with respect to the component. See EUROMOD

Functions - The policy function BenCalc for a description of the function *BenCalc* and its parameters.

Parameters are grouped via the *Grp/No* column. This column contains the same (integer) number for all parameters belonging to the same group. By convention groups are in ascending order, i.e. group 1 followed by group 2, etc. However, this is not imperative, i.e. group 4711 followed by group 1147 would work as well. Note that the *Band_* parameters of the

function *SchedCalc* form an exception

to this rule, as in this case the group number also determines the order of the bands. See EUROMOD Functions - The policy function SchedCalc for a description of the function *SchedCalc* and its parameters.

Footnote parameters

Footnote parameters are parameters, which provide further

information on other parameters or parts of other parameters (symbolically: they contain footnote information to something in the "text"). For example the footnote parameter *#_LowLim* may define a lower limit for a formula or a part of a formula.

The footnote parameter and the operand it belongs to are

linked by an integer number. For the footnote parameter this number is contained in the *Grp/No* column, whereas the

operand is simply followed by *#number*.

For example, consider the formula *yem**#2** **

10%, and the footnote parameter *#_LowLim*, whose *Grp/No* column is set to **2**. If the parameter *#_LowLim* takes a value of 1000

(indicated in the *System* column(s))

the formula would be calculated as 10 % of employment income (i.e. *yem*), if employment income is higher than 1000 and as 10% of 1000, if employment income is up to 1000.

Note that a footnote parameter can be linked to more than

one operand, however only within the same function.

For more information on footnote parameters see EUROMOD Functions - Footnote parameters for the further specification of operands.

Parameters with editable parameter name

Some functions comprise parameters, where the *Policy* column does not contain a parameter name in the narrow sense, but information that could be seen as a parameter value, which is equal for all systems. For example, for the function *DefConst* the *Policy* column contains the name of the constant, whereas the system columns contain the value of the constant for the respective system. In principle this is a shortcut - it would also be possible to use two parameters (*Const_Name*, *Const_Value*) - with the

advantage of being clearer and using less space. For obvious reasons the *Policy* column is editable for such parameters. For more information see the descriptions of the functions *DefConst*, *DefVar*, *Uprate*, *SetDefault*, *DefIL*.

Adding Parameters

The *Add Parameter* Form

Open the *Add Parameter Form* to add a parameter to a function. The form is opened by right clicking the respective function to open its context menu which provides the menu item *Show Add Parameter Form*. Alternatively the form can be opened by selecting the function and pressing *Control-A*. The dialog is non-modal, which means it can be kept open while working on other things. Accordingly it can be resized, shifted and minimised - except for the latter it stays on top of all windows.

The form shows all parameters that can be added to the function (in the column *Parameter*) with a description (in the column *Description*). That means, amongst others, that "normal" parameters, which are already part of the function are not listed (e.g. the parameter *TAX_UNIT* or the parameter *formula* of the function *ArithOp*). Excepted from this rule are "special" parameters, i.e. parameters, which can be added more than once (e.g. the parameter *Var* of the function *DefOutput* or the parameter *Comp_Cond* of the function *BenCalc*) and parameters, which have "aliases" (see Replacing an "alias" parameter below).

In consideration of its non-modality the form observes what the user is doing and adapts its content respectively. If a function or one of its parameters is selected, the content adjusts to this function. If a policy is selected, the content is cleared (as the form would not know which function of the policy it should refer to).

Two options allow for a possibly more manageable content of the form:

- ***Show Common Parameters*:** If this option is unchecked, the form shows only parameters, which are specific to the respective function, i.e. it hides parameters, which are common to all (or most) functions, like *TAX_UNIT*, *Output_Var*, *Who_Must_Be_Elig*, etc.
- ***Show Footnote Parameters*:** If this option is unchecked, the form hides footnote parameters (parameters starting with a #). For detailed information see EUROMOD Functions - Footnote parameters for the further specification of operands.

To assess help on the function displayed by the ***Add Parameter Form*** use the buttons

- ***Description***: leading to a descriptive explanation of the function, usually with a couple of examples. Alternatively press the ***F5 key***.
- ***Summary***: leading to a full (but brief) description of the parameters of the function. Alternatively press the ***F6 key***.
- For information on the handling of the form press the ***F1 key***, in fact leading to the page you are looking at.

Note that pressing the *Escape* key closes the form (without any consequences) unless an editable field in the table has the focus.

Adding parameters

To add one or more parameters to the function select them by ticking the corresponding check boxes. Then click the *Add* button (the button with the green plus).

You can also use the keyboard to add parameters. Tick/untick the corresponding check boxes: first use the arrow keys (*up*, *down*, *left*, *right*) to move the focus to their row in the *Add* column, then press the *Space* key. Press the *Enter* key to add the selected parameters. Note that pressing the *Enter* key always adds the selected parameters, unless an editable field in the table (e.g. *Count*) has the focus.

Note that if in the main view a parameter is selected, new parameters are added after this parameter, whereas if a function is selected, new parameters are added at the end of the function.

Adding parameters, which allow for more than one incidence

Parameters, which allow for more than one incidence (e.g. the parameter *Var* of the function *DefOutput*), can be added in bulk. To add X incidences of a parameter to the function, put X the *Count* column of the parameter, which automatically selects the parameter (i.e. its check box is ticked). Press the *Add* button or the *Enter* key to add the parameters to the function.

Adding group parameters

For group parameters (e.g. the parameters *Comp_Cond* and *Comp_PerTU* of function *BenCalc*) the form suggests the next available group number in the *Grp/No* column. The group number can be changed, though this is not recommended, as it should not be necessary. If group parameters are added in bulk (see above) the adding process assigns the group number indicated in the *Grp/No* column to the first incidence and automatically augments the group number for each further incidence.

See Working with EUROMOD - Presentation of (normal and) special parameters for more information on group parameters.

Replacing an "alias" parameter

As mentioned above, parameters having an "alias" (e.g. the parameter *Output_Var* with its alias *Output_Add_Var*) are still listed even if they are already part of the function. More precisely they are listed in the column *Replaces* while the column *Parameter* contains their alias. Once the *Add* button (or the *Enter* key) is pressed the parameter is replaced by its alias. The replaced parameter keeps its value: If for example *Output_Var* was set to the variable *bun_s*, *Output_Add_Var* will still be set to this variable.

Note that if both aliases (e.g. as well *Output_Var* as *Output_Add_Var*) are removed from the function, they both appear in the *Add Parameter Form's* list. However, trying to add them both leads to an error message.

Adding the parameter [Placeholder]

For some functions (*DefConst*, *DefVar*, *Uprate*, *SetDefault*, *DefIL*) the form contains the special parameter *[Placeholder]*. If this parameter is added, its *Policy* column is editable (while the *Policy* column is read only for any other parameter). This allows the user to replace *[Placeholder]* by an appropriate value (usually the name of a variable). The *Policy* column stays editable after replacing *[Placeholder]*, to allow for later changes of the value. Also see Working with EUROMOD - Presentation of (normal and) special parameters.

Note that the parameter *[Placeholder]* can always be added in bulk (see Adding parameters, which allow for more than one incidence above).

Changing parameters

In general parameter values can be changed by typing the new value in the respective cell. However, depending on the parameter's value type, the user interface provides different input assistance. The following concentrates on this input assistance, for detailed information on parameters' value types see EUROMOD Functions - Types of parameter values.

Editing categorical parameters

For categorical parameters, i.e. parameters, which take a certain limited choice of values, the user interface provides lists from which the required value can be selected. Examples are taxunit parameters (the user interface lists the available assessment units), yes/no parameters (the user interface lists the values *yes* and *no*), incomelist parameters (the user interface lists the available incomelists), the parameter *Who_Must_Be_Elig* (the user interface lists the values *nobody, one, one_adult, all, all_adults*), etc. Note that the lists always contain the value *n/a*.

There are two ways of using these lists without applying the mouse. The first is to type the initial letters of the required value, which selects the first entry in the list that starts with these letters. For example, if the list contains the values *tu_household_sl*, *tu_individual_sl* and *tu_family_sl*, typing *tu_* (or *t* or *tu*) selects *tu_household_sl*, while typing *tu_i* selects *tu_individual_sl*. For the second way, firstly select the respective cell (using the *Up*, *Down*, *Left*, *Right* keys) and make it editable by pressing the *F2* or the *Enter* key, then press the *Page Down* key. This opens the list. Use the *Up*, *Down* keys to select a value and confirm with the *Enter* or *Tab* key or close the list with the *Escape* key.

Editing formula, condition and variable parameters

For formula parameters, i.e. parameters, taking formulas like *yem*10%+ils_ben/2*, condition parameters, i.e. parameters taking conditions like *{IsDependentChild & dag<3}* and variable/incomelist parameters, i.e. parameters taking the names of variables and incomelists, the user interface provides assistance in the form of so called "IntelliSense".

This mechanism observes what the user types and makes "suggestions" within a small window. For example, if the user is editing a formula parameter and typing an *i*, the mechanism provides a list of operands, which are allowed in a formula (i.e. variables, incomelists, queries, etc.) and start with an *i* (e.g. *idhh, idperson, ..., ils_earns, il_bsaBase, ..., IsMarried, IsParent, ...*). If the next character typed is an *l*, the list is reduced to operands starting with *il* and so on. Thus the user can either continue typing or accept a suggestion by selecting it with the mouse. Alternative to the mouse the *Up, Down* keys can be used for selecting and the *Tab* key for confirming the selection. The *Enter* key also confirms a selection but at the same time stops formula editing by moving the focus to the next row. Note that if the *Up, Down* keys are used for selection, IntelliSense displays a short description of the operand. Moreover, note that small icons help distinguishing the operands by showing different pictures for variables, incomelists, queries, etc.

Note that, if one types the character 'q', IntelliSense lists all EUROMOD queries and thus provides an overview over all available queries. Moreover note that typing the character '<' lists the operators *<max>, <min>* and *<abs>()*.

User interface's assistance in adding and using footnote parameters

Some operands of formula and condition parameters allow for so called footnote parameters. To learn more about this feature please consult EUROMOD Functions - Footnote parameters for the further specification of operands and Working with EUROMOD - Presentation of (normal and) special parameters. This section concentrates on the user interface's support for generating such information.

Generating footnote parameters: To equip an operand in a formula with a *footnote*, type # after the operands name, e.g. *yem#*. Thereupon IntelliSense displays a list containing entries like *#x1[_LowLim], #x2[_LowLim], #x3[_LowLim], ..., #x1[_Level] , #x2[_Level] ...*. To define for example a lower limit for the operand select *#x1[_LowLim]*, which at first results in *yem#x1[_LowLim]*. However, once you complete formula editing (e.g. by pressing the *Enter* key) *#x1[_LowLim]* is replaced by *#1*. Moreover, the footnote parameter *#_LowLim*, with *Grp/No* set to 1, is added. The only task left in finalising the definition of the operand's lower limit is setting *#_LowLim's* value to the respective amount.

To be precise, the footnote number is not always set to 1, but to the next free number, i.e. if the function already has footnote parameters with the numbers 1, 2 and 3, the number of the new footnote parameter is set to 4.

The purpose of *#x2[_LowLim]*, *#x3[_LowLim]*, etc., as offered by IntelliSense, is to allow for generating different lower limits within one formula editing procedure. You may for example produce the following formula *yem#x1[_LowLim]* + *yse#x2[_LowLim]* + *poa#x3[_LowLim]*. Once you complete formula editing the formula is adapted to *yem#1* + *yse#2* + *poa#3* (provided the function did not yet contain any footnote parameter). Moreover, three footnote parameters *#_LowLim*, with *Grp/No* set to 1, 2 and 3, are added.

Note that it is also possible to generate footnote parameters by using the *Add Parameter Form* (see Working with EUROMOD - Adding parameters).

Assigning existing footnote parameters: The fact that footnote parameters can be assigned to more than one operand, as well as the possibility to generate footnote parameters by using the *Add Parameter Form*, opens the questions how existing footnote parameters can be assigned to an operand. To do so, again type *#* after the operands name, e.g. *yem#*. If there are existing footnote parameters, IntelliSense displays them in the form *#_LowLim1*, *#_Level3*, etc. Selecting e.g. *#_Level3* results in *yem#3* and thus the existing footnote parameter #_Level, with *Grp/No* set to 3, is assigned to the operand *yem*. Of course you can simply type *yem#3* with the same effect, the possibility of using IntelliSense for this purpose is more or less just a reminder, that existing footnote parameters can be reused.

Generating #_amount parameters: *#_amount* parameters are in fact just special footnote parameters, whereupon the speciality is based in the fact that the only operand they can be assigned to is the operand *Amount*. Accordingly they are generate as any other footnote parameter with the difference that, instead of *operand#*, you must type *a* (or *am*) to bring IntelliSense to list *Amount#x1*, *Amount#x2* and *Amount#x3*. The rest of the procedure is as described above.

Assigning existing #_amount parameters: Again there is no difference to assigning other footnote parameters, except that, instead of *operand#*, you must type *a* (or *am*) to bring IntelliSense to list *Amount1*, ..., *Amount4711* etc.

User interface's assistance in adding query parameters

Some EUROMOD queries provide (optional or compulsory) query parameters to specify their behaviour. These query parameters are very similar to footnote parameters. Just like footnote parameters query parameters start with #_ (e.g. *#_AgeMin*, *#_AgeMax*) and they are assigned to the respective query by *#number*. To learn more about EUROMOD *queries* please consult EUROMOD

Functions - Queries. Again this section concentrates on the user interface's support for generating query parameters.

IntelliSense lists queries with (optional or compulsory) query parameters followed by *#x*, e.g. *nPersInUnit#x*. Once formula editing is completed *#x* is replaced by the next free footnote number, e.g. *nPersInUnit#4*, if the function already contains three *footnote* or query parameters. Moreover, the respective query parameters are added, with *Grp/No* set to the free footnote number. In our example the query parameters *#_AgeMin* and *#_AgeMax* are added, with *Grp/No* set to 4 (for both).

If you want to avoid the generation of query parameters, for example because the parameters are optional and you do not want to define them, just delete the *#x* after the name of the query. This informs the user interface that generating query parameters is unwanted.

Replacing parameter names by their 'aliases'

A left-click on a parameter name (in the *Policy* column) may show a dialog that asks whether the parameter should be replaced by its 'alias'. The term 'alias' refers to parameters, which replace each other, for example *Comp_perTU* / *Com_perElig* or *Output_Var* / *Output_Add_Var* (each also vice versa). Confirming the question with *OK* replaces the parameter’s name respectively.

Changing the order of parameters

There are two ways to change the order of parameters: by using (a) drag and drop or (b) key *Up* / key *Down*. For each of them first select the respective parameters. A single parameter is selected by simply clicking it (more precisely, its name in the *Policy* column). A range of parameters is selected by selecting the first parameter (i.e. its name in the *Policy* column), then pressing and holding the *Shift* key while selecting the last parameter. Light blue back colour shows the selection. For a more detailed description of how to select cells see Working with EUROMOD – Selecting components and values.

For (a) click the respective parameters with the left mouse button, drag them with pressed mouse button to the new position and drop them there by releasing the mouse button.

For (b) press the *Control* as well as the *Up* key to move the respective parameters up, respectively the *Control* as well as the *Down* key to move them down.

Note that parameters can only be shifted (up or down) within the function they belong to. That means, this way it is not possible to move a parameter from one function to another.

Deleting (ranges of) parameter values

Note that this subsection describes how to delete parameter *values* and not how to delete parameters *as a whole*. For the latter please refer to Working with EUROMOD – Deleting parameters.

A parameter's value is 'deleted' if the *Delete*-key is pressed while the respective cell is selected. 'Delete' is put in quotation marks because in fact the value is not set to empty but to *n/a* (not available or not applicable). Note that this does not apply to parameter values in the narrow sense only, but also to the switch of a policy or function.

It is also possible to delete (more precisely set to *n/a*) a range of parameter values by selecting the respective range before pressing the *Delete*-key. A range of parameter values is selected by selecting the first parameter value (i.e. the respective cell), then pressing and holding the *Shift* key while selecting the last parameter value. Light blue back colour shows the selection. For a more detailed description of how to select cells see Working with EUROMOD – Selecting components and values. Note that, in order to delete parameter *values*, the *Policy* column must not be part of the selection as this instructs the interface to delete the whole parameter.

Automatic commenting of incomelist components

The function context menu shows an additional menu item if opened for a *ILDef* function: *Description as Comment*. If the menu point is selected the user interface puts a description of all components of the concerned incomelist into the *Comments* column. As an example, for the Estonian incomelist *il_UAB_meanstest* consisting of the components *ils_origy*, *bmact* and *bcclg_s*, the user interface will describe these components as *original income*, *parental benefit (vanemapalk)* and *large family parent allowance (seitsme- ja enamalapselise pere vanema toetus)*. Note that country specific descriptions are used where available.

Deleting parameters

Parameters can be deleted by either using the context menu or via the *Delete*-key. In both cases first select the parameter(s) you want to delete. Then press the *Delete*-key. Alternatively right click the name of (one of) the parameter(s) for opening the context menu and selecting the menu item *Delete Parameter(s)*.

A single parameter is selected by simply clicking it (more precisely, its name in the *Policy* column). A range of parameters is selected by selecting the first parameter (i.e. its name in the *Policy* column), then pressing and holding the *Shift* key while selecting the last parameter. Light blue back colour shows the selection. For a more detailed description of how to select cells see Working with EUROMOD – Selecting components and values.

Note that the selection needs to include the *Policy* column (i.e. the name(s) of the parameter(s)). If only system columns and/or the group or comment columns are selected, the interface assumes that the respective values should be deleted – see Working with EUROMOD – Changing parameters (paragraph *Deleting (ranges of) parameter values*). Moreover, if you use the context menu, you need to open it via a parameter contained in the selection, otherwise not the selected parameters are deleted, but the parameter that belongs to the context menu. In addition note that with the context menu it is only possible to delete parameters belonging to one function – use the *Delete*-key if you want to delete a wider range of components.

A security question is issued before the parameters are finally removed. Note that you can opt to not show the security query anymore - see Working with EUROMOD - Managing warnings.

Copying parameter values (and comments)

There are several options to copy the values of one or more parameters to other parameters. They are summarised below. Please note that these options refer to the content of the parameter tree, i.e. they do not generate new rows. You may want to check for options to copy whole sets of parameters in Working with EUROMOD – Copying policies and Working with EUROMOD – Copying functions.

Copying from and to the clipboard

This mechanism can be applied within a country and between countries. Moreover, information can be transferred from and to "outside", e.g. Excel.

Please note that the mechanism works without restrictions with the system columns (displaying parameter values and policy or function switches[1]) and the *Comment* column (displaying comments to policies, functions or parameters). However there are certain restrictions concerning the *Policy* and *Grp/No* column. While all cells of these two columns can be copied and their values be transferred e.g. to Excel, not all cells can be edited by pasting. The *Grp/No* column is only editable for parameters, thus pasted values assigned to policy or function rows are omitted. The edit ability of the *Policy* column is even more restricted, only a few specific cells are editable, e.g. containing the components of incomelists (function *DefIL*) or the names of constants/variables (functions *DefConst* and *DefVar*). As for the *Grp/No* column, only those editable cells can be changed by pasting.

- **Selecting the copy region:** Select the region you want to copy by selecting the first cell of the region, and then pressing and holding the *Shift* key while selecting the last cell of the range. Light blue back colour shows the selection. For a more detailed description of how to select cells see Working with EUROMOD – Selecting components and values. If no region is selected, the copy operation refers to the focused cell.
- **Copying:** Press *Ctrl-C* or *Ctrl-Insert* to copy the selected region. Alternatively use the right mouse button to click within the selection. This opens the context menu providing, amongst others, the menu item *Copy Value(s)*.

- **Selecting the paste region:** Select the top-left cell of the region where you want to paste the copied values. If you want to restrict the paste operation to a certain region, select this region as described above.
- **Pasting:** Press *Ctrl-V* or *Shift-Insert* to paste the content of the copied region to the region selected for paste. Alternatively use the right mouse button to click within the selection. This opens the context menu providing, amongst others, the menu item *Paste Value(s)*.
  Please note that, if the copied region refers to one single cell, while the paste region is a selection of several cells, all cells of the paste region are filled by the value of the copied cell.
  Also note that it may happen that the copied region is too large or too small (because the paste region is a selection as well or the cell selected for starting the paste operation is too far to the bottom or right). In this case only the cells for which there is a matching cell can be filled.

Spreading parameters over systems

A quite common task is to spread parameter values from one system to (all) other systems. For example, if one implements a new function, one will set the parameter values firstly in one system and then, for a start, copy the values to all other systems to finally adapt the values as required.

The user interface supports this task by the following mechanism: Select the parameters you want to spread as described in Working with EUROMOD - Selecting components and values. For spreading a single parameter value just focus the respective cell.[2] Pressing *Alt-S* spreads the selected parameter values to all other systems.

Note that parameters are only spread to visible rows and columns (systems). This allows limiting the spreading to selected systems (see Working with EUROMOD - The column chooser for information on hiding systems). Moreover, be aware that functions of collapsed policies as well as parameters of collapsed functions are not involved in the spreading mechanism (see Working with EUROMOD - Expanding and collapsing policies and functions). Also hidden rows are excluded (see Working with EUROMOD - Hiding policies, functions and parameters).

Finally, note that the mechanism checks the selection for not exceeding the boundaries of a system. That means if parameters of several systems are selected, pressing *Alt-S* is ignored.

Setting whole functions / policies to n/a

There are two options to set the switch of a policy or function to *n/a*: *n/a* and *n/a (all components)*. Selecting the former sets the respective policy or function to *n/a*, whereas the latter sets all functions and parameters contained in the policy, respectively all parameters contained in the function, to *n/a* as well.

[1] Note that if the values pasted to policy or function switches are no switch values (i.e. on, off, toggle, n/a) the user interface displays an empty cell, however the pasted values are actually stored in the XML parameter file, thus the executable will issue a respective error message (e.g. reporting that 'grumlmumpf' is not a valid switch). The same is true for any cell displaying a drop down list when being edited (e.g. for the parameters *TAX_UNIT*, *who_must_be_elig*, etc.).

[2] The focused cell shows a red doted border. You focus the cell by e.g. clicking into it. To close any possibly opening editor, use the Escape key. Alternatively to the mouse you can use the arrow keys.

Defining Uprating Factors

The user interface provides support for defining uprating

factors for monetary variables based on an index table. An example illustrates best what this means. Let’s assume we have the following index table:

Index	Reference	2012	2013	2014	Comment
Harmonised CPI (index 2012=100)	$Factor_CPI	100	104.3	110.2	Source: Eurostat
Average monthly salary	$Factor_WAGE	150.4	155.8	159.4	Source: Eurostat

On the basis of this table the user

interface is able to generate definitions of uprating factors for each available dataset and tax-benefit system, which can then be used to uprate monetary variables (e.g. in an *Uprate* function).

More concretely, assume the country comes with two datasets,

cc_2012_a1 and *cc_2013_a1*, and three systems, *cc_2012*, *cc_2013* and *cc_2014*. The user interface will define the following uprating factors:

For system *cc_2012*:

- *$Factor_CPI = 1* if dataset *cc_2012_a1* is used
- *$Factor_WAGE = 1* if dataset *cc_2012_a1* is used
- *$Factor_CPI = 0.9588* (= 100 / 104.3) if dataset *cc_2013_a1* is used
- *$Factor_WAGE = 0.9653* (= 150.4 / 155.8) if dataset *cc_2013_a1* is used

For system *cc_2013*:

- *$Factor_CPI = 1.043* (= 104.3 / 100) if dataset *cc_2012_a1* is used
- *$Factor_WAGE = 1.0359* (= 155.8 / 150.4) if dataset *cc_2012_a1* is used
- *$Factor_CPI = 1* if dataset *cc_2013_a1* is used
- *$Factor_WAGE = 1* if dataset *cc_2013_a1* is used

For system *cc_2014*:

- *$Factor_CPI = 1.102* (= 110.2 / 100) if dataset *cc_2012_a1* is used
- *$Factor_WAGE = 1.0598* (= 159.4 / 150.4) if dataset *cc_2012_a1* is used
- *$Factor_CPI = 1.0566* (= 110.2 / 104.3) if dataset *cc_2013_a1* is used
- *$Factor_WAGE = 1.0231* (= 159.4 / 155.8) if dataset *cc_2013_a1* is used

These factor definitions can then be used for example in an *Uprate* function as follows:

Policy	**cc_2012**	**cc_2013**	**cc_2014**	*Comments*
Uprate	**on**	**on**	**on**	
bch	$Factor_CPI	$Factor_CPI	$Factor_CPI	child benefits are uprated by CPI
yem	$Factor_WAGE	$Factor_WAGE	$Factor_WAGE	employment income is uprated by wage index

which means, for example, that the variable *bch* is uprated with a factor 1.102 for the system *cc_2014*, if dataset *cc_2012_a1* is used. If instead dataset *cc_2013_a1* is used, *bch* is uprated with a factor 1.0566.

Generating the index table

To achieve the above described, the developer firstly must fill in the index table. This is accomplished via a dialog opened by clicking the button *Uprating Indices* in the ribbon *Country Tools*. The tab *Raw Indices* provides the respective index table.

To add indices (i.e. rows) to the table just fill the last, empty row respectively. This will automatically generate another empty row for a further new index.

To delete an index select the respective row by clicking its

column-header (the whole line will be marked by blue background). Then press the *Delete* key. For deleting several

indices use the *Ctrl* and *Shift* keys as usual to select the rows.

Then press *Delete*.

To add years (i.e. columns) to the table, fill the year in

the field above the button *Add Year* to then press the button. Use the button *Delete*

Year to delete the year selected in the combo-box above the button. Year columns can be shifted by using the mouse to drag them to the desired location.

In order to paste values into the index table (e.g. from

Excel), firstly select the respective cells. Then use *Ctrl-V* (or *Shift-Insert*) for pasting. Similarly, to copy values from the table to the clipboard, firstly select the respective cells, to then use *Ctrl-C* (or *Ctrl-Insert*) for copying.

Pressing the button *OK* will - based on the information in the index table - generate the definitions of uprating factors as exemplified above.

Note that, if an index is set to zero for one or more years this may, on using it, have the following effects:
If the zero-index refers to the system year this leads to "uprating by zero", i.e. setting the concerned variable(s) to zero.
If the zero-index refers to the data year this leads to "division by zero", i.e. setting the concerned variable(s) to NaN (not a number).
In both cases the programme issues a respective warning.

Checking

factors per dataset and system years

The tab *Factors per*

Data and System allows for controlling, which factors will be generated, on pressing the *OK* button. Once a

dataset is selected in the combo-box on top of the dialog, the table will show for this dataset the factors for each system. Note that obviously factors can only be calculated, if the (income) year of the dataset (see Working with EUROMOD - Configuring datasets) is available and the *Raw*

Indices table provides as well the year of the dataset as the year of the system. If this is not the case, the table shows *n/a* for not available.

The button *Update* can be used to update the content of the control table, if new information was entered to the *Raw Indices* table.

Using

the factor definitions

Most likely the factor definitions will be used in *Uprate* functions as outlined above. However, in fact the factors can in general be used like constants (defined by *DefConst* functions).

The only particularity is, that the constants will

adopt different values dependent on the used dataset. The interested user may read the following paragraph to learn how this works, respectively for a closer understanding.

Important! Please note, that if a new dataset is added to an *Uprate* function (by the parameter *dataset*)

the factor definitions will not be automatically available for this dataset. In fact it needs opening and closing the dialog for defining Uprating Indices, as the respective background information is only generated by this action. Reading the following paragraph on the technical background may make this clearer.

Checking

the usage of the factors

The button *Check Usage* allows for assessing which variables are uprated by any factor. Clicking the button opens a dialog which explores all *Uprate* functions (switched to *on* or *toggle*) and collects the variables for which the factors are applied. Moreover, it checks if any factor is applied for the parameter *Def_Factor*.

If a factor is applied on a variable for only part of the systems (i.e. tax-benefit-years), this is indicated by showing the respective years in parenthesis behind the variable. It is also possible to restrict the check to a range of (or single) tax-benefit-years by choosing the respective start- and end-year in the *From-to-*boxes. In this context please

note that the check takes only standard systems into account, i.e. systems named *cc_yyyy*.

Please also note that the check is limited to *Uprate* functions. It does not check if the factor is used anywhere else in the implementation of the country. To provide for such an extended search, press the button *Go to Component Use*, which opens the Component Use dialog.

Technical background

The information displayed by the *Raw Indices* table, as well as the factor definitions generated by the user interface on the base of this table, are

stored in a hidden policy.[1] This policy comprises one *DefVar* function -

this function contains the information stored in the *Raw Indices* table[2].

Moreover, it contains one *DefConst* function for each of the country's datasets.

The *DefConst* functions contain a definition for each factor of the *Raw Indices* table, in

the form:

Policy	*Grp/No*	**cc_2012**	**cc_2013**	**cc_2014**
DefConst		**on**	**on**	**on**
$Factor_CPI	1	1	1.043	1.102
const_dataset	1	cc_2012_a1	cc_2012_a1	cc_2012_a1
$Factor_WAGE	2	1	1.0359	1.0598
const_dataset	2	cc_2012_a1	cc_2012_a1	cc_2012_a1
DefConst		**on**	**on**	**on**
$Factor_CPI	1	0.9588	1	1.0566
const_dataset	1	cc_2013_a1	cc_2013_a1	cc_2013_a1
$Factor_WAGE	2	0.9653	1	1.0231
const_dataset	2	cc_2013_a1	cc_2013_a1	cc_2013_a1

As can be read in the description

of the function *DefConst*, the parameter const_dataset effects,

that: "... constant is only defined if the respective dataset is used for the run". In the example that means that for the system *cc_2012* the constant *$Factor_CPI* takes on the value 1, if run with the dataset *cc_2012_a1* and 0.9588, if run with the dataset *cc_2013_a1*.

Note that the "IntelliSense" (see Working with EUROMOD - Changing parameters, paragraph *Editing formula, condition and*

variable parameters) will for the system *cc_2012* show two entries for the constant *$Factor_CPI* (i.e. *$Factor_CPI (1)* and *$Factor_CPI (0.9588)*). As "IntelliSense" is only typing assistance it does not matter which of it is selected.

[1] The name of the policy is *DefUpratingFactors_cc* and it is the very first policy in the spine.

[2] The first parameter of this function stores the *Raw Indices* tabel's years (in the form "*2012°2013°2014*"). Each further parameter contains the information for one index, i.e. raw of the *Raw*

Indices table (in the form "*Harmonised*

CPI (index 2012=100) °$Factor_CPI°100°104.3°110.2°Source: Eurostat") The information is stored in the parameters' name while the parameters' values are all set to *n/a* (to avoid unnecessary redundancies). The function is switched off and thus just used for informing the dialog.

Undo and redo

To undo the last change click the blue arrow pointing to the left in the top left corner of the user interface or press *Ctrl-Z*. This procedure can be repeated until there is no change left to be undone.

To redo the last undone change click the blue arrow pointing to the right in the top left corner of the user interface or press *Ctrl-Y*. This procedure can be repeated until there is no change left to be redone.

Note that "change" may encompass several operations. If for example several parameters are added by clicking *Add* in the *Add Parameter Form* (see Working with EUROMOD - Adding parameters), undo will not remove each parameter separately, but all parameters added with the same *Add* action at once.

Expanding and collapsing policies and functions

In order to view all the functions in a policy (or all the parameters in a function respectively) you can expand the policy/function in the following ways:

- by clicking

  the little *right arrow* button on the

  left side of its name.
- by focusing

  on that policy/function and pressing the *Ctrl + Right Arrow* key combination on your keyboard.
- by focusing

  on that policy/function and pressing the *plus* key on your keyboard (this will also work for multiple selected policies/functions).
- by focusing

  on that policy/function and pressing the *multiply* key on your keyboard (this will fully expand the focused/selected items and all its children).
- by right-clicking on the policy/function/parameter name and selecting the menu item *Expand All Functions* (this will fully expand the focused/selected policies and all its children).

Similarly, you can return to the compact view by collapsing an expanded policy/function in the following ways:

- by clicking

  the little *down arrow* button on the

  left side of its name.
- by focusing

  on that policy/function and pressing the *Ctrl + Left Arrow* key combination on your keyboard (if the focused item is already collapsed, this will move the focus to the item's parent).

- by focusing on

  that policy/function and pressing the *minus* key on your keyboard (this will also work for multiple selected policies/functions).
- by focusing

  on that policy/function and pressing the *divide* key on your keyboard (this will fully collapse the focused/selected items and all its children).
- by right-clicking on the policy/function/parameter name and selecting the menu item *Collapse All Functions* (this will fully collapse the focused/selected policies and all its children).

Finally, to view all functions and parameters of all policies (i.e. the whole available information) right click the header of the *Policy* column and select the menu item *Expand All Policies*. To return to the compact view, i.e. hide all functions and parameters, right click the header of the *Policy* column and select the menu

item *Collapse All Policies*.

Hiding & showing parts of the Spine

Hiding policies, functions and parameters

The row context menu, opened by right clicking on the row number, allows for hiding and unhiding (ranges of) policies, functions and parameters.

Hiding a single policy, function or parameter

Right click on the number of the respective row and select the menu item *Hide Row* from the row context menu. For example, click row number 3 for hiding the 3rd policy in the spine*;* row number 4.2.6 for hiding the 6th parameter of the second function of the fourth policy in the spine, etc. Alternatively select the row of the policy, function or parameter and press *Alt-H*.

Hiding multiple selected policies, functions or parameters

First select a range of cells, then right click on *any* row number and select the menu item *Hide Selected Rows* from the row context menu. This will hide all the policy, function and parameter lines that are currently selected. Note that if there is no current selection, then this menu item will appear disabled.

Hiding all but a single policy, function or parameter

Right click on the number of the respective row and select the menu item *Hide all other Rows* from the row context menu. For example, click row number 3 for hiding all policies except the 3rd policy in the spine*;* row number 4.2.6 for hiding all parameters of the second function of the fourth policy in the spine except the 6th, etc. Alternatively select the row of the policy, function or parameter and press *Alt-O*.

Hiding a range of policies, functions or parameters

Right click on the first row of the range to open the row context menu and move the mouse over the menu item *Hide Rows unto*. This opens a sub menu containing a combo box with a list of row numbers. Select the last row of the range to hide all rows within the range.

Note that the listed row numbers are limited to rows, which firstly follow the first row of the range and secondly are on the same level as the first row. For example, if row number 3 is selected as the first row, i.e. the 3rd policy in the spine, the list of row numbers comprises 4, 5, 6, 7, etc., i.e. all policies after the 3rd. Selecting 6 hides policies 3, 4, 5 and 6 and obviously also the included functions and parameters. If row number 4.2.6 is selected as the first row, i.e. the 6th parameter of the second function of the fourth policy in the spine, the list of row numbers comprises 4.2.7, 4.2.8, 4.2.9, etc., i.e. all parameters after the 6th parameter of the second function of the fourth policy. Selecting 4.2.8 hides parameters 4.2.6, 4.2.7 and 4.2.8.

Hiding policies or functions that are "n/a" in all currently visible systems

Right click on any row number and select the menu item *Hide all "n/a" Policy/Function rows*. This will go through the whole spine and hide all policies and functions which are set to "n/a" in all of the currently visible systems. Note that this only works for policies and functions; it will not affect parameters even if they are set to "n/a" for all visible systems.

Hiding selected policies or functions that are "n/a" in all currently visible systems

First select a range of cells, then right click on any row number and select the menu item *Hide selected "n/a" Policy/Function rows*. This will go through the selected policies and functions and hide those which are set to "n/a" in all of the currently visible systems. For example you may want to quickly hide "n/a" functions within a given policy. In this case you would click on that policy and then right click on its row number and select the *Hide selected "n/a" Policy/Function rows* item. Note that this only works for policies and functions; it will not affect parameters even if they are set to "n/a" for all visible systems.

Unhiding policies, functions or parameters

Right clicking any row and selecting the menu item *Unhide Rows* from the row context menu opens a dialog that lists any hidden ranges of rows. Check the box of the range you want to unhide and press the button *Unhide*. You can also use the "Check All" checkbox at the bottom to quickly check/uncheck all the available ranges. Note that nested hiding is not taken into account. For example, if you first hide functions 3.2 to 3.6 and afterwards policies 2 to 4, the dialog only lists policies 2 to 4 and unhides all functions and parameters included.

Private comments

In addition to the comment, which is presented in the rightmost column, it is possible to record "private comments". Such comments are private in the sense that they are not available in public versions (see Working with EUROMOD – Generating a EUROMOD public version).

Recording, changing and deleting private comments

To record or change a private comment right-click the *Comment* column of the respective policy, function or parameter to open the context menu and select the menu item *Private Comment*.[1] This opens an input box for editing the comment. The private comment is deleted by clearing it in the input box.

Display of private comments

The existence of a private comment is indicated by a small red rectangle in the comments cell. The respective text can be viewed (like a tool tip) by moving the mouse over the cell.

[1] In fact the context menu can also be opened by clicking any other column except the *Policy* column. (The policy context menu does not provide the menu item to avoid further overload.)

Changing text size

The bottom right corner of the user interface contains the field *Textsize:* + -. Click + to increase and - to decrease the font size.

Saving changes

To save your changes open the main menu (above the button *Run EUROMOD*) and select the menu item *Save*. Alternatively press *Ctrl-S*. For further information on file organisation and structure see EUROMOD Installation and File Structure.

Saving a copy of a country

To generate a copy of a country select the menu item *Save As* from the main menu (above the *Run EUROMOD* button). This opens a dialog allowing indicating the *Long Name* and the *Short Name* of the copied country. The former is the countries full name (e.g. Hungary, Spain, MySpain, etc.), the latter is the country code (e.g. HU, ES, MS, etc.). Note that the country code needs to be unique, i.e. no other country with this code exists yet. Moreover, you are asked to indicate a *Flag* in portable network graphic format (png). Ideally the size of the image is 28x17 pixels. You can also select the image with the file search dialog opened by pressing the button alongside the *Flag* field. If you do not indicate a flag image, the flag of the origin country will be used.

Once the specifications are confirmed with *OK* the copy of the country is displayed in the country gallery (ribbon *Countries*) and can be loaded and edited as any other country. Note that, after *Save As*, the loaded country is the copied. The origin country is not loaded anymore and has the status of the last saving before the *Save As*.

For information on the files, which are generated by the copy process see EUROMOD Installation and Architecture.

Auto-saving

The user interface provides an auto-saving feature, which automatically saves changes to a country's parameter files in configurable intervals of time. The changes are not saved in the original country's parameter files but in temporary files called *astmp_cc.xml* and *astmp_cc_DataConfig.xml* and stored in the temporary folder (see EUROMOD Installation and File Structure). If the user interface fails to load a country it suggests using these files if existent.

To configure the auto-saving interval, or to en- or disable auto-saving, select the menu item *Configuration* from the main menu (above the *Run EUROMOD* button). The tab *Auto Save* of the appearing dialog allows for defining these settings.

File locking

In order to avoid that several developers change a country at the same time, the user interface uses a locking mechanism.[1] That means once a developer opens a country "normally", i.e. in read-write modes, any further developer who opens the same country, gets an is-in-use-message, which tells who is using the country and asks whether the country should be opened in read-only modus. The user interface allows for three reactions to this message: Firstly, with pressing 'Yes' the country is opened in read-only modus, meaning that saving is disabled. Secondly, with pressing 'Cancel' the developer decides to not open the country. Thirdly, with pressing 'No', the country is opened "normally", i.e. in read-write modes. This last option is foreseen for cases, where due to an unforeseen event the locking of the country was not removed after closing it, i.e. locking is irrelevant. Developers should however be aware that the option allows for actually overwriting other developers changes or getting once own changes overwritten - it must therefore be used with care.

Please note that file locking is not only applied on countries but also on add-ons and on the variables file.

[1] Technically the user interface generates a small text file alongside the parameter files, which contains information on who uses the country. The file is deleted once the country is closed again.

Changing Countries' Settings

Changing country settings

To view or change a country's settings select the ribbon *Country Tools* and click the button *Country* in the *Configuration* group.

The dialog allows for changing the *Long Name* of the country (e.g. Austria, Greece, …).

Moreover it is possible to set the country "private". For further information concerning private components see Working with EUROMOD - Generating a EUROMOD public version.

Note that the *Short Name* of the country cannot be changed by the user. It needs to correspond to the filenames of the XML-files storing the country (see EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Format of country parameter files).

Changing system settings

To view or change the settings of a country's systems select the ribbon *Country Tools* and click the button *Systems* in the *Configuration* group. This opens a dialog, which lists all systems with their settings and allows for changing the settings. For more information on EUROMOD systems see EUROMOD Basic Concepts - Presentation of countries' tax-benefit systems and EUROMOD Basic Concepts - Terminology.

The settings include:

Exchange Rate: This setting indicates the rate used by the model to convert national currency into Euro or vice versa (i.e. Euro amounts are multiplied by the rate to get amounts in national currency). For any country whose national currency is the Euro the parameter should be set to 1.

Currency Parameters: This setting indicates the currency used for monetary parameter values. Possible values are *euro* or *national* (standing for national currencies like UK Pound, Danish Krone, Polish Zloty, etc.). For more information on EUROMOD parameters see EUROMOD Basic Concepts - Presentation of countries' tax-benefit systems.

Currency Output: This setting indicates the currency used for generating EUROMOD output. Possible values are again *euro* or *national*. For more information on EUROMOD output see EUROMOD Basic Concepts - EUROMOD input and output).

Private: This setting, if checked, indicates that the system is not (yet) ready to be presented to the public. In practise the only impact of the setting concerns the generation of a EUROMOD public version, see Working with EUROMOD - Generating a EUROMOD public version.

Income for Unit Head Definition: This setting indicates which income concept is used as a default for determining which member of an assessment unit is the "head". A combo box provides all incomelists available for the system, to allow for selecting an appropriate income concept. The default setting is *ils_OrigY*. For further information consult the description of the function *DefTU* in EUROMOD Functions - The system functions DefTU and UpdateTU.

Click *OK* to confirm any changes or *Cancel* to close the dialog without any consequences. Note that changes are only definite once the country is saved.

Before that you can still use the undo functionality (see Working with EUROMOD - Undo and redo) or close the country without saving.

Configuring datasets

To view or change the settings of the datasets that can be used to simulate the country's tax-benefit systems select the ribbon *Country Tools* and click the button *Databases* in the *Configuration* group. This opens the *Configure Databases* dialog.

Assigning datasets to systems

The upper part of the dialog shows a table where the row headers list all datasets available for the country, while the column headers list all available systems. The intersection of a dataset (row) and system (column) indicates whether the system can be run with the dataset, in other words whether they form a so-called system-dataset combination (see EUROMOD Basic Concepts - Terminology and Working with EUROMOD - Running EUROMOD for more information).

There are three possible settings. A cross (x) indicates that the dataset and the system form a system-dataset combination. *best* also denotes a system-dataset combination, however in addition this combination is a *Best Match* (see EUROMOD Basic Concepts - Terminology). *n/a* means that the system cannot be run with the dataset.

Note that, if a system is copied (see Working with EUROMOD - Adding systems), the datasets assigned to the original system are automatically also assigned to the copied system. Use the *Configure Databases* dialog to change this, if necessary.

A right click in the datasets table opens a context menu, which allows for more convenient assigning of datasets to systems, by allowing to set the same value (x, or n/a) for all system-dataset combinations of a specific system or a specific dataset.

Adding removing or renaming a dataset

To add a dataset click the *Add Dataset* button, which opens a file search dialog allowing for the search of a text file containing data suitable to run (one or more of) the country's systems.

To delete a dataset, select it and click the *Delete Dataset* button. Note that the dataset is removed without any further warning, but you can still undo this action by closing the dialog with the *Cancel* button or by using the undo functionality (see Working with EUROMOD - Undo and redo). Of course the dataset (text file) is not deleted physically, the removal concerns only the ability to use the dataset with the country's systems.

To rename a dataset, select it and click the *Rename Dataset* button. This opens a textbox, which allows for entering the new name.

The settings of the selected dataset

Below the *Datasets / Systems* table the dialog shows the settings of the selected dataset.

Collection Year: This setting indicates the year the data were collected.

Income Year: This setting indicates the year the monetary values within the data refer to.

Currency: This setting indicates in which currency data are stored. Possible values are *euro* or *national* (standing for national currencies like UK Pound, Danish Krone, Polish Zloty, etc.).

Decimal Sign: This setting indicates whether data uses point (.) as decimal sign or comma (,).

Path: This setting indicates a specific path to locate the dataset. Usually it is left empty, to instruct EUROMOD to locate the dataset at the default path (see Working with EUROMOD - Open project).

Private: This setting, if checked, indicates that the dataset is not (yet) ready to be presented to the public. In practise the only impact of the setting concerns the generation of a EUROMOD public version, see Working with EUROMOD - Generating a EUROMOD public version.

Use Common Default: Checking this option means that any variable not existent in data, but used by the system, is set to zero, i.e. no error message is issued. In this context also consider the role of the function *SetDefault*.

String Output Variables: This setting indicates a list of variables, separated by space, which exists in the described data and are to be transferred to output.
The values of the variables may be alpha-numeric (i.e. contain strings).
Note that a warning is issued, if the variables are not found. Moreover, note that the name is not case-sensitive.

Read Expenditure-related Variables: Checking this option means that the programme reads all expenditure-related variables, where expenditure-related

means that the name of the variable must start with x (for expenditure), p (for price) or q (for quantity). All other characters must be digits (0 to 9).
These variables are then disposable, independent on whether they are declared in the variables file or not. That means amongst others that they are covered by using e.g. VarGroup=x* in the DefOutput function (independent on whether they are used elsewhere).

Click *OK* to confirm any changes or *Cancel* to close the dialog without any consequences. Note that changes are only definite once the country is saved. Before that you can still use the undo functionality (see Working with EUROMOD - Undo and redo) or close the country without saving.

Finding an error

An error message of the EUROMOD executable has about the following format:

```
Error:        Variable or incomelist 'borer' not defined.
System:       at_2011
Policy:       tin_at; row 7
Function:     arithop; row 7.3
Parameter:    formula; row 7.3.2
Value:        borer
Identifier:   b10e103f-0652-4a77-b2ec-d021636b98f4
```

The error can be found in the user interface by either looking for the indicated row numbers or by searching the parameter or function by its identifier. The *Search by Identifier* dialog serves the latter approach. To open the dialog, select the ribbon *Country Tools* and press the button *Search by ID* in the *Search* group. Copy the *Identifier* (b10e103f-0652-4a77-b2ec-d021636b98f4 in the example) from the error message to the field *Identifier* and press *Find* to jump to the component.

Note that the dialog does not support finding an error, which was produced by running a system with an add-on (see Working with EUROMOD - Running EUROMOD paragraph *Running add-on systems*). In the case of add-on-systems the row numbers indicated by the error message refer to the order in which components are actually processed. This order is generated at run-time and therefore not visible "from outside". Similar is true for the identifier, as it is internally generated by the user interface and not visible from outside either.

Some comments on EUROMOD's error reporting

EUROMOD distinguishes two phases of error handling and two (actually three) types of errors. Error handling phase one comprises reading and checking parameters and reading data – let's refer to it as **read-time** – while phase two concerns computing the necessary calculations and outputting results – let's refer to it as **run-time**. The two types of errors are **critical** and **non-critical errors** (the latter also referred to as **warnings**).

During read-time the model gathers all errors, independent of type, without stopping. Once the phase is finalised it always stops if there are critical errors, to issue respective error messages. If there are only non-critical errors it only stops if the box *Do not stop on non-critical errors* in the *Advanced Options* of the *Run EUROMOD* dialog is unticked (see Working with EUROMOD – Running EUROMOD paragraph *Advanced settings*), otherwise – and of course if there are no errors – it continues. During run-time EUROMOD always stops immediately once a critical error occurs. Whether it stops on non-critical errors again depends on ticking *Do not stop on non-critical errors*: if not ticked it stops immediately, if ticked it continues and issues the gathered error messages after finishing the phase or once a critical error occurs. Finally, above a third type of error was mentioned – this concerns "very critical" errors, which necessitate an immediate stop, independent of the phase.

Concluding, it may be informative to learn, which errors are classified as critical respectively non-critical by the model. Intuitively speaking one could say that EUROMOD is very strict during read-time and classifies most errors as critical. In contrast the model is rather lenient during run-time and considers errors as non-critical, as long as it finds a way to continue its calculations (warnings inform about the respective handling in their error message). The motivation for this is found in the fact that phase two takes much longer than phase one, i.e. the model tries to immediately inform the users after the short read-time that something went wrong, while they may confidently go for a coffee during run-time and will get a summary of errors once they are back. In reality running one system of a country (with a not too big dataset) usually takes a few minutes, which is unfortunately too short for the coffee. However, one could imagine a reform that comprises a lot of countries, systems or loops. In that case the user may first get the parameters right and have time for her coffee, while running this big reform.

Searching and replacing

Selecting the ribbon *Country Tools* and pressing the button *Search + Replace* opens the *Search and Replace* dialog. The dialog can also be accessed by pressing *Ctrl-F* to open it in the search mode and by *Ctrl-H* to open it in the replace mode.

Searching

To search for example for the string *abc*, type *abc* into the field *Search* and press either the button *Search Next* (or use the key-combination *Alt-N*) or the button *Search Previous* (key-combination *Alt-P*). This finds the cell nearest to the currently focused cell, which contains the string *abc*. *Search Next* finds the nearest cell after and *Search Previous* finds the nearest cell before the focused cell. "Finding" means that the respective cell is focused (red-dotted border) and made visible by expanding the parent nodes, if necessary, and scrolling to the position of the cell.

It is possible to use search patterns. That means *?* can be used for symbolising one arbitrary character and * for symbolising any number of arbitrary characters. *bch??_s*, for example, finds *bch00_s*, *bch01_s*, *bchba_s*, ..., *bchyc_s*, while *bch*_s* finds the listed strings as well as *bch_s*, *bchba01_s*, ..., *bchucrg_s*.

Note that any searched string or pattern is stored by the dialog, in the sense that one can pick it from the list, which is displayed by clicking the arrow button right of the *Search* field.

Specifying the search

The *Search and Replace* dialog offers several options to specify the search:

Search in ... Cells

- As a default the search refers to all cells. Note however, that cells which are hidden due to row-hiding (see Working with EUROMOD - Hiding policies, functions and parameters) or because the system was moved to the *Hidden Systems Box* (see Working with EUROMOD - The Hidden Systems Box) are not even found if the option *All Cells* is selected.
- Selecting the option *Visible Cells* restricts the search to cells which are visible in the sense that they are not hidden because their parent node is

collapsed. Note that cells hidden due to scrolling are still found (in fact the interface scrolls to the position of the found cell).

- Selecting the option *Selected Cells* restricts the search to selected cells (see Working with EUROMOD - Selecting components and values).

Search in ... Columns

- As a default the search refers to all columns, except those moved to the *Hidden Systems Box* (compare above).
- Selecting the option *System Columns* effects that the search only takes system columns into account.
- Selecting the option *Policy Column* effects that the search only takes the *Policy* column into account.
- Selecting the option *Comment Column* effects that the search only takes the *Comment* column into account.

Search by ...

As a default (option *Search by Columns*) the search first finds all occurrences of the searched string or pattern in the first (affected) column, to then find all occurrences in the second column, etc. Selecting the option *Search by Rows* effects that the search first finds all occurrences in the first (affected) row, to then find all occurrences in the second row, etc.

Match Case

As a default the search is not case sensitive. Ticking the box *Match Case* enforces case sensitivity.

Match Entire Cell Content

As a default the searched string or pattern does not need to be the only content of the found cells. For example *yem* finds a cell containing *yem* as well as a cell containing *yem*10%*. Ticking the box *Match Entire Cell Content* effects that only cells, which contain only the searched string or pattern are found (i.e. the first match in the example).

Include Private Comments

As a default the search does not include private comments (see Working with EUROMOD – Private comments). Ticking the box extends the search to jump to the respective cell in the *Comment* column, if the private comment matches the search criteria. Note that the box is not available in replace mode.

Replacing

Pressing the button *Replace* ... displays the field *Replace by* and the button *Replace All*. The field *Replace by* allows for indicating a string by which the string in the field *Search* is replaced. To start (single) replacing, press *Search Next* to find the first match.[1] Then press *Replace* to replace *Search* by *Replace by* in the found cell. This moves the focus automatically to the next matching cell (where *Replace* can be pressed again).

Pressing the button *Replace All* replaces all occurrences of *Search* by *Replace by*, taking into account the following restrictions (which also apply to single replacement).

- Replacing does not allow for search patterns, i.e. *?* and * cannot be used.
- Matches in the *Policy* and *Grp/No* columns can only be replaced if the respective cell is editable, otherwise an error message is displayed. Please refer to Working with EUROMOD - Copying parameter values (and comments) paragraph *Copying from and to the clipboard* for restrictions on replacing values in the *Policy* and *Grp/No* column.
- The search options as described above, in paragraph *Specifying the search* are valid for replacing as well.

The *Search and Replace* dialog does not provide a *Search All* option, however a similar functionality is provided by the *Component Use* dialog with the option *Component named* (see Working with EUROMOD - Checking component use).

[1] Note that you need to start the replacing by focusing a cell that matches *Search* - otherwise an error message is issued.

Checking component use

Selecting the ribbon *Country Tools* and pressing the button *Component Use* opens the *Component Use* dialog. This dialog allows for checking which "components" are used for the implementation of the country in EUROMOD. The following components can be included into the check:

Included components

Variables: If this option is selected all EUROMOD variables, as listed in the variables file, are included into the check. As the checking for the use of all variables may cause that the check takes quite a lot of time, the variables can be restricted to those *having country specific description*. For closer information see Working with EUROMOD - Administration of EUROMOD variables. Moreover, the check-boxes *monetary*, *non-monetary*, *simulated* and *non-simulated* allow for further restriction to the variables which feature the respective characteristics.

Assessment Units: Selecting this option includes assessment units in the check (see EUROMOD Basic Concepts - EUROMOD terminology).

Incomelists: Selecting this option includes incomelists in the check (see EUROMOD Basic Concepts - EUROMOD terminology).

Queries: Selecting this option includes EUROMOD queries in the check (see EUROMOD Functions - Queries).

Component named: This option allows for checking the use of a specific component, i.e. a specific variable, assessment unit, incomelist or query. Moreover, it allows for checking the use of variables and constants generated with the functions *DefVar* and *DefConst* (which are not taken into account by the option *All Variables*).

Ignore if Switched Off: Selecting this option means that components solely used in switched off policies respectively functions are not recognised as used.

Include Systems: The list allows for restricting the check for usage to the selected systems.

Listing of used components

Pressing the *Start* button starts the check. A progress bar is displayed to inform about the time still necessary for the check. Once the check is finished, components as selected are listed if they are used in the implementation of the country. If a component is used several times, it is listed as often as it is used, each time with the respective location of the use.

Jumping to a component

Selecting a specific use of a component and pressing the button *Goto* closes the dialog to jump to the selected use in the country's implementation. If the dialog is reopened (by pressing the button *Component Use*) it still shows the content of the last check, i.e. jumping to a component does not "destroy" the results of the check. The results of the last check are in general only cleared by a new check, but not for example by closing the dialog.

Storing results

Pressing the *Store* button allows for specifying a file where the current listing of component usage is stored in tabulated text format.

Administrating policy switches

The content of this article is partly out-dated and thus supplemented and updated by the article Extensions (fka Policy Switches).

The EUROMOD user interface provides a special way to switch policies on or off via the run dialog (instead of having to change parameters). To learn how this works from the user's point of view please see *Working with EUROMOD - Running EUROMOD*, paragraph *Using policy switches*. The present section concentrates on describing how to administrate the underlying information.

Implementing a switchable policy involves three steps, where step one and two are interchangeable in order. Firstly, the respective policy must be implemented in all countries where this policy is available. Secondly, the user interface must be "informed" about the existence of the (new) switchable policy. Finally, default values (per system and dataset) must be defined for the switches. Step one is usual developing work and requires no further description at this place.

Step two: Defining the switchable policies

The button *Administrate Switchable Policies* in the ribbon *Administration Tools* opens a dialog, which provides options for adding, deleting and changing switchable policies. The information needed by the user interface for a switchable policy has two components:

- the policy's name as used in the countries' implementation must be entered in the column *Name Pattern*. The information is in fact treated as a "search pattern". That means *?* can be used for one arbitrary character and * for any number of arbitrary characters. *TCA??_??*, for example, would cover *TCA_at*, *TCA_uk*, *TCA1_cy*, *TCA12_ee* ... This implies that one policy switch can be used for a group of policies.
- a description (e.g. *Tax Compliance Adjustments*) must be entered in the column *Long Name*. This information is used for displaying in the run dialog.

Please note that the user interface issues a warning, if existing[1] switchable policies are deleted or if their name pattern is changed, which says that "switches in the country files are not automatically updated" and suggests updating them by opening the *Set Policy Switches* dialog and closing it with *OK*. What this means is explained below in the paragraph *Update of policy switches in the country file*.

Step three: Defining default values for the switches

means that for all available system-dataset-combinations of each country a default value for the switch of the policy must be defined. This is accomplished via a dialog activated by the button *Policy Switches* in the ribbon *Country Tools*. The left part of the dialog provides a list of existing switchable policies as entered in step one. Please note that only such switchable policies are displayed, which are implemented for the country. For example, if the policy *TCA_hu* is not implemented, the dialog does not display this switchable policy for Hungary, even if a switchable policy with name pattern *TCA_??* exists.

Checking a switchable policy of this list fills the table in the right part of the dialog with the respective switches for each system-dataset combination. An empty (and not changeable) cell indicates that the respective system-dataset combination is not available (i.e. the dataset cannot be used to run the system). For the (other) switches three values are possible:

- ***on*** means that the policy is switched on as a default for this system-dataset-combination.
- ***off*** means that the policy is switched off as a default for this system-dataset-combination.
- ***n/a*** means that the policy is not switchable[2] for this system-dataset-combination. This informs the user interface to disable the respective button in the run dialog (more precisely, set its caption to blank). Note that, if for a switch no explicit default value is specified, *n/a* is used as a default.

These default values are applied if the policy-switch-columns in the run dialog are not displayed. Moreover, they are the *default* values for the policy-switch-buttons (see *Working with EUROMOD - Running EUROMOD*).

A right click in the switches table opens a context menu, which allows for more convenient definition of the switch values, by allowing to set the same value (*on*, *off* or *n/a*) for all system-dataset combinations of a specific system, a specific dataset or simply for all.

Update of policy switches in the country files

Once default values of switchable policies are confirmed by closing the dialog with *OK*, the switches of the respective policies in the country are automatically changed to *switch*. Moreover, those switches are not changeable by users.[3] Please note that policy switches are able to reflect system-specificity, but they are not able to reflect system-dataset-combination-specificity (which is actually the level on which policy switches are defined). For this reason a policy's switch for a specific system shows *switch* as soon as the policy is switchable for this system for at least one dataset.

As mentioned above (see last sentence of paragraph *Step two: Defining the switchable policies*) switches are not automatically updated if a switchable policy is deleted or its name pattern is changed. For example, consider a switchable policy with the name pattern *Random_??*. If this policy is defined for a country and default values are specified, its switches are set to *switch*. If now the name pattern (for whatever reason) is changed to *Rand_??* in the *Administrate Switchable Policies* dialog, the switches will still show *switch*. But in fact the policy is no longer switchable (as not matching the name pattern anymore) but completely switched off. To correct this, please proceed as the warning described above suggests: open the *Set Policy Switches* dialog (which will not anymore display the switchable policy *Rand_??*, as no such policy is defined for the country) and close it with *OK*. This updates the switches in the country.[4]

Passing the information to the executable and treatment

The user interface passes the switches of a switchable policy (or a group of such policies) to the executable via the configuration file (see EUROMOD Installation and Architecture - EUROMOD software (user interface and executable) - The configuration file). Note that the output header file (see *Working with EUROMOD - Running EUROMOD*, paragraph *Running the selected system-dataset combinations*) is extended by a column captioned *Policy Switches*. This column shows, for each system, information like *bta_??=on;tca_??=off*.

Automatic renaming of the output file based on non-default policy switch values

In the *Set Policy Switches* dialog you will also find a checkbox that allows you to automatically rename the output file based on non-default policy switch values. If this checkbox is checked, and if the output file name defined in the spine follows the default format (e.g. *xx_YYYY_std*), then it can be automatically renamed to signify which switches have been changed in the *Run* dialog. Specifically, for each switch that has a non-default value, the appropriate text will be appended at the end of the filename, while the *_std* suffix is removed (e.g. *ee_2010_std* can become *ee_2010_btaoff_fyaon*).

[1] Existing, in this context, only means that the switchable variable was not added in this session of the dialog. The user interface does not check whether the switchability via this policy is actually applied somewhere in the model.

[2] The policy can be not switchable for two reasons: Firstly, the system-dataset-combination may not allow respective calculations, in other words it is not available. Secondly, the policy is not switchable for this specific system-dataset-combination. In the latter case the switch in the country file applies, which may be set (as usual) to *on*, *off* or *toggle*.

[3] To make them changeable again, one needs to reset the default switches in the *Policy Switches* dialog to *n/a*.

[4] The reason for this a bit unhandy approach is technical simplicity - a technical complex solution seems not to be justified (necessary) given the rather low probability of the event.

Extensions (aka Policy Switches)

The content of this article supplementes and updates the partly out-dated article Administrating policy switches.

Extensions are an advancement of **Switchable Policies** in the following sense:

- Instead of one policy, they can comprise several policies (single) functions and (single) parameters.
- In addition to the **Global Switchable Policies** (**Extensions** from now on), **Country Specific Extensions** are introduced.
- The elements (policies/functions/parameters) belonging to an **Extension** can be either switched on or off. This is explained in more detail in the next paragraph.
- **Extensions** allow for the same actions as *Groups* (i.e. *Set Visible, Set Not Visible, Expand*) and two additional actions: *Set Private, Set Not Private*.
- Similar to *Groups* **Extensions** are indicated by little coloured symbols in the row-number column of the spine.
  While **Groups** use boxes as markers, **Extensions** use ticks (symobl for on) and crosses (symbol for off).

Switching Extensions *on* or *inactive*

In general **Extensions** can still be switched on and off like the current **Switchable Policies** (i.e. in the *Extension Switches dialog* for defaults and the *Run-tool* for the current run). But the advancement described in the 3rd point above makes it more useful to talk about *on* and *inactive* instead of *on* and *off*.
This may be best illustrated by an example:
Assume an **Extension** called BSA_Alt which comprises the policies BSA_Alt_cc and BSA_cc.
When the **Extension** is on BSA_Alt_cc should be on and (the base-line policy) BSA_cc should be off.
When the **Extension** is inactive BSA_Alt_cc should be off and BSA_cc should be not affected (i.e. usually on).

Adding Extensions

Global Extensions are (still) administrated via the button in the *Administration Tools*-tab (which was renamed from *Administrate Switchable Policies* to *Admin Global*).
Country Specific Extensions are administrated via a button in the *Country Tools*-tab. The dialog opened by this button is the same as for adding *Groups*.

Adding policies, functions and parameters to Extensions

Similar to *Groups*, one can add elements to **Extensions** either via the context-menu or via buttons in the *Country Tools* tab. But different to **Groups** the item *Add to* is split into *Add to, switch on* and *Add to, switch off*. The paragraph above (*Switching Extensions on or inactive*) hopefully explains these two buttons.

To show the difference between *on*-elements and *off*-elements in the spine, the former are displayed with filled symbols and the latter with empty symbols.

Extension Actions

As mentioned above, **Extensions** share the actions *Set Visible*, *Set Not Visible* and *Expand* with *Groups*. In addition they allow for:

Set Private: all elements of the **Extension** are set private.

Set Not Private: removes the private-attribute from all elements of the **Extension**.

Note that there are two such sets of buttons: one in the *Country Tool*-tab and one in the *Administration Tools*-tab. The latter shows only **Global Extensions** and sets the selected **Extension** private for all countries. The former shows all **Extensions** and is intended to only set the **Extension** private for the respective country. However, to avoid misunderstandings, there is a request whether the user wants to perform the action for all countries, if the **Extension** is global.
To be precise there is third set of the buttons in the context-menu. This has the same functionality as the buttons in the *Country Tool*-tab.

It may be useful to know that the intention of this private (un)setting is to allow for customised Releases: One could release a, in general private, **Extension** to a selected group of people by using *Set Not Private* for this **Extension** before hitting the *Generate Public Version* button. Then one can use *Set Private* to undo the "publication" for further Releases.

Formatting

Conditional Formatting

To access the *Conditional Formatting* dialog select the ribbon *Display* and click the button *Conditional Formatting*. The dialog supports two ways of conditional formatting:

Conditional Formatting

The upper part of the dialog allows for adding, removing and setting the characteristics of conditional formats (in the following short CF). CF's are formats, here text and background colour, which are applied to a parameter value, if it fulfils the CF's condition.

Adding a CF: Click the *Add* button (the button with the green plus) to add an undefined CF, i.e. a row in the list of CFs.

Removing a CF: Select the row of the CF and click the *Delete* button (the button with the red cross) to remove it from the list.

The CF's condition defines the criteria the parameter value must fulfil. It has the form "_{pattern} OR {pattern} OR {pattern}_" etc. That means, to fulfil the condition, the parameter value must match one of the patterns between the curly brackets, where a pattern consists of characters and the wildcards * (for zero or more arbitrary characters) and ? (for one arbitrary character). For example the condition "_{*#m*} OR {*#y*}_" is fulfilled by parameter values like "_200**#m**_" or "_50**#y**+yem*10%_". Please note that the user interface tries to interpret the condition as far as possible and ignores faulty parts. As an example, the faulty condition "_{*#m*} OR {*#y*_" would be recognised as "_{*#m*}_", i.e. the second incorrect part is ignored (note the missing second curly bracket).

Systems to apply the CF on: The column *Systems to Apply* in the CF’s row indicates the systems the CF should be applied on. To change this setting, click the respective cell with the left mouse button. In the appearing dialog select the respective systems and press *OK* to overtake them into the table.

The CF's text and back colour: The columns *Back Color* and *Text Color* in the CF’s row define the respective colour settings of the CF. To change them, click the respective cell with the left mouse button. In the appearing colour dialog select the desired colour and press *OK* to overtake it into the table. To set the colour back (i.e. use no special back or text colour for the CF), right click the respective cell and select *Clear Color*.

Differences to Base System Formatting

The lower part of the dialog allows for highlighting differences of a system's parameter values in comparison to the parameter values of another system. It lists all available systems (in the column *System*) and, if available, the system to compare with, i.e. the base systems (ergo listed in the column *Base System*). The base system is defined by a click in the respective cell, which opens a dialog that allows for selecting an appropriate system. A text and/or back colour can be defined to accomplish the formatting, in the same way as described above for conditional formatting. In order to maintain a certain consistency over countries, the button *Restore Default Formatting*, allows for setting the (back and text) colours for all system/base system pairs back to a common default.

You also have the option to expand any differences between a System and its Base System, by ticking the box in the column *Expand Differences*. This will make all differences "immediately visible" by expanding any functions and policies that contain different values, and it is independent to the formatting (colours) you applied.

It is recommended to use text colour formats for CF and back colour formats for highlighting base system differences (or vice versa), if applied on the same system, to avoid conflicting formats.

Click *OK* to confirm any changes or *Cancel* to close the dialog without any consequences. Note that changes are only definite once the country is saved. Before that you can still use the undo functionality (see Working with EUROMOD - Undo and redo) or close the country without saving.

Automatically setting the Base System Formatting

You can access this functionality from the ribbon *Display* by clicking the button *Automatic Conditional Formatting*. This function will try to match all the Systems according to their names and automatically select the appropriate Base System for each one (but will only do so if a Base System does not already exist for this System). For these Systems that it was able to find and assign a Base System, it will also set the formatting to the Default colours. Note that this function is only able to match standardized System names, such as "*CC_YYYY*" (where "CC" is the short country code and "YYYY" is the 4-digit year) or "*CC_YYYY_SomethingElse*" (where "SomethingElse" can be anything at all). For the system UK_2012 the function will for example try to find Base System

UK_2011, for the system UK_2012_reform it will try to find Base System UK_2012. If the function cannot find a match for a given System, then it will not change anything.

Groups

A **Group** is a collection of policies, (single) functions and (single) parameters. This is a purely visual feature which allows for marking items to be connected in a certain way, but has no effect on the model run.

Groups are characterised by **Group Markers**: little coloured symbols in the row-number column of the spine. In addition to the visual aspect, they allow for actions like show, hide or expand.

Administration and application of **Groups** is carried out via a respective group of buttons in the *Display*-tab and/or via the context menu.

Adding Groups

To add a **Group** click the button *Administrate* in the *Groups* section of the *Display*-tab. This opens a dialog allowing for adding, changing and deleting **Groups**. For defining the look of the **Group** click the *Look* column of the **Group's** row.

Adding policies, functions and/or parameters to Groups

There are two possibilities: either use the *Add to* button in the *Groups* section of the *Display*-tab, or use the *Groups/Add to* subitem of the context menu. Note that all ***selected*** policies/functions/parameters are added to the **Group**. Also note that the adding also concerns all sub-items, i.e. if a policy belongs to a **Group**, all (current and prospective) functions and parameters of the policy automatically belong to the **Group** – likewise for functions. Finally note that only those **Groups** are listed, where the concerned item(s) not already belong(s) to.

Removing policies, functions and/or parameters from Groups

There are two possibilities: either use the *Remove from* button in the *Groups* section of the *Display*-tab, or use the *Groups/Remove from* subitem of the context menu. Note that all ***selected*** policies/functions/parameters are removed from the **Group**. Also note that only those **Groups** are listed, where the concerned item(s) (currently) belong(s) to.

Group Actions

There are two possibilities to activate any group action: either use the buttons *Set*

Visible, *Set Not Visible* and *Expand* in the *Groups* section of the *Display*-tab, or use the respective subitems of the context menu.

Set Visible: all elements of the **Group** are shown in the sense that a possible hidden-state is removed, however, the parent-elements are not automatically expanded.

Set Not Visible: all elements of the **Group** are hidden.

Expand: all elements of the **Group** are shown (possible hidden-state removed) and actually made visible by expanding.

Note that these actions are ignorant in the sense of not caring about any current or contradicting state: *Set Visible* does for example not care if elements were actually hidden before. Similarly, if one first sets Group A visible and then Group B not visible, with an element belonging to both groups, the element will be not visible in the end. Also note that the actions are in principle not permanent, i.e. get lost once a country is closed. The visible-state is however part of the *view settings*, which can be stored (see option in the *Project Configuration* dialog).

Marking nodes with colour

Select the rows of the policies, functions and/or parameters you want to equip with a background colour. A single row is selected by simply clicking it. A range of rows is selected by selecting the first row, then pressing and holding the *Shift* key while selecting the last row. Light blue back colour shows the selection. For a more detailed description of how to select cells see Working with EUROMOD – Selecting components and values. Then select the ribbon *Display* and choose the desired colour in the *Marking* group. To clear the colour of a group of policies, functions and/or parameters, select the respective rows, to then click the button *Clear* in the *Marking* group. To clear any colour, click the button *Clear All* in the *Marking* group.

Note that, to avoid conflicting formats, the colour is only applied to the *Policy*, *Grp/No* and *Comment* columns, but not to the system columns (see Working with EUROMOD - Conditional formatting).

Setting bookmarks

Setting a bookmark

Select the row of the policy, function or parameter you want to bookmark, select the ribbon *Display* and click the button *Set Bookmark*. Entering a name in the appearing dialog and confirming with *OK* adds the bookmark alongside the *Undo/Redo* buttons in the top left corner of the user interface.

Moving the mouse over the bookmark shows its name. Moreover, the symbol displayed indicates whether the bookmark refers to a policy (symbolised by a blue ball), function (symbolised by the function symbol) or parameter (symbolised by a star).

Applying a bookmark

Clicking the bookmark sets the focus on the bookmarked row and, if necessary, makes the row visible by expanding the policy/function it belongs to.

Removing a bookmark

Right click the bookmark you want to remove and select the menu item *Remove from Quick Access Toolbar*.

Note that bookmarks are personal settings, i.e. they are not reflected in the country's parameter files and therefore not shared with other users. This implicates that saving has no effect on bookmarks and undo is not available. Use the approach described above to remove bookmarks.

Administration of EUROMOD projects

A EUROMOD project is actually not much more than a notation of a folder containing the EUROMOD file structure (see EUROMOD Installation and Architecture - EUROMOD content (parameter files)). In fact, the user interface always points to such a folder and its main task is to allow for visualising, administrating and editing its content. In other words the user interface always refers to a concrete EUROMOD project.

In practical terms this means that a user can easily switch from one project to another. For example she can have one project containing the " EUROMOD core project", i.e. the one she gets from the EUROMOD team. Then she may have a project where she implements changes for the purpose of analysing certain policy changes. Then she may have another project where she implements changes for the purpose of analysing other policy changes, etc. Switching between the projects is accomplished via the item *Open Project* of the main menu. [1] See Working with EUROMOD - Administration of EUROMOD projects - Open project for further information.

The user also may want to generate a new project. This is described under Working with EUROMOD - Administration of EUROMOD projects - New project.

Except from describing a certain EUROMOD file structure, a project also has some (project specific) features. For example, it may have a specific *Input Folder*, meaning that it points to other datasets as e.g. the EUROMOD core project. Another quite important feature is whether a project is version-controlled or not. See Working with EUROMOD - Administration of EUROMOD projects - Configure project for further information.

[1] You find the main menu above the *Run EUROMOD* button. Press the little arrow to open it.

Generate a new EUROMOD project

The item *New Project* of the main menu[1] (which can also be found in the Version Control ribbon in the *PROJECT* group) opens a dialog which allows for generating a new EUROMOD project. For a description of what a EUROMOD project is see Working with EUROMOD - Administration of EUROMOD projects.

The user interface requires the following information for generating a EUROMOD project:

Project Path and Project Name

The *Project Path* is the path where the new project is to be stored. The user interface will generate a folder using the name you provided in the *Project Name* field, containing the EUROMOD file structure at the indicated path. Note that the folder named in *Project Path* must already exist and that it must not contain a folder *Project Name* already. For more information on the EUROMOD file structure see EUROMOD Installation and Architecture - EUROMOD content (parameter files).

Base Project

In fact, a EUROMOD project will rarely be generated "from scratch", usually it will be a copy of an existing project, that then can be adapted. Consequently, the generate-project-dialog allows for selecting this *Base Project*, with the following options:

- *Project on Disk*: With this option selected, the dialog allows for selecting (or typing) the name of the folder, where the base project is stored[2]. Note that this will be the default choice if you are not logged into Version Control.
- *Project on VC*: With this option selected, the dialog allows for selecting a base project stored in the EUROMOD version control system. Once the option is ticked, the combo-box alongside will list all the VC projects you have access to.
Note however, that this option is only available if the user is connected to the EUROMOD version control system (in which case, it is actually the pre-selected option). Consequently, if this is not (yet) the case, clicking the

option *Project on VC* opens the version control login dialog, asking for a user name and password (see EUROMOD Version Control - Logging in and out). Also note that, when first ticking the option, it may take a short while, as the available projects must be requested form the VC-system. For further information on the EUROMOD VC-system see EUROMOD Version Control).

- *No Base Project*: As mentioned above, projects will rarely be generated "from scratch", but it is still possible and enabled by selecting this option.

Content of the new project

The button *Define Content* allows for specifying which parts of the base project are overtaken into the new project. Consequently, it is not available if the option *No Base Project* is selected. Moreover, an error message will be displayed on clicking it, if no base project was selected yet.
Otherwise clicking the button opens a dialog which lists all units (countries, add-ons, config files etc.) of the base project. The user can then tick the units she wants to overtake into the new project.

[1] You find the main menu above the *Run EUROMOD* button. Press the little arrow to open it.

[2] At the local disk or some network drive available to the user.

Open a EUROMOD project

The item *Open Project* of the main menu[1] opens a dialog which allows for switching the EUROMOD project to which the user interface refers to. For a description of what a EUROMOD project is see Working with EUROMOD - Administration of EUROMOD projects.

The dialog offers two ways to select the project:

- Selecting a project that was opened by the user interface before
- Selecting a project that was not yet opened by the user interface, but is stored on the user's disk[2]

Selecting a project that was opened before

For this purpose, select the respective project from the list of available projects displayed by the combo-box *Project Folder*. Note that the user interface "knows" about the available (i.e. at least once opened) projects, because it stores "user settings" (in fact project settings) for each of them. These user settings contain amongst others the project's path.

Selecting a project that was not yet opened

For this purpose, click the button alongside the field *Project Folder* and select the respective path with the help of the opened file-dialog. If you know it by heart you can also type the project's path.
Note that the selected folder must contain the EUROMOD file structure (see EUROMOD Installation and Architecture - EUROMOD content (parameter files)).

Opening the user interface for the very first time

There is in fact an alternative instance when a user becomes to see the open-project-dialog than via the menu-item *Open Project*. That is, when the EUROMOD user interface is opened for the very first time.[3] In this case the dialog shows an additional button *New* ..., which allows for opening the new-project-dialog (see Working with EUROMOD - New project) and thus, instead of opening an existing project, generating a new one.

[1] You find the main menu above the *Run EUROMOD* button. Press the little arrow to open it.

[2] ... or some other path the user has access to (e.g. a network drive).

[3] ... or if the user-settings (i.e. the information that is stored internally about the user's personal specifications) are invalid. The most likely reason for invalid user-settings is that they refer to a project folder that does not exist (any more) or does not contain the EUROMOD file structure.

Configuration dialog

The configuration dialog, which is opened by selecting the main menu's item *Configuration*, offers options to configure the project currently loaded by the user interface. There are four tabs, which provide the following options.

Tab General

Output Folder

This setting instructs EUROMOD to generate, as a default, any output at the indicated path. Type the respective path into the field or select it with the folder search dialog opened by the button alongside the field. Note that this default output path can still be changed for a particular run in the respective field of the run dialog (see Working with EUROMOD - Running EUROMOD paragraph *Selecting the output path*).

Input Folder

This setting instructs EUROMOD to look, as a default, for input data at the indicated path. Type the respective path into the field or select it with the folder search dialog opened by the button alongside the field. Note that this default input path can still be changed for a particular dataset in the *Configure Databases* dialog (see Working with EUROMOD - Configuring datasets).

Set Standard Input/Output-Paths

Pressing this button sets the *Output Folder* to ***ProjectPath****\Output* and the *Input Folder* to ***ProjectPath****\Input*.

Close User Interface with Last Country

If this option is ticked the user interface is closed, once the last country (and add-on) is closed. If the option is not ticked, the user interface is not closed in this case, but displays the EUROMOD logo (as it does when it is opened and no country is yet loaded).

Tab Version Control

Log In At Project Load

If this option is checked, the user is automatically connected to version control when the project is loaded, provided that the project is version controlled. For further information see EUROMOD Version Control - Logging in and out.

Time-Out

This field indicates the time in milliseconds any single version control operation (essentially upload or download of a unit) may take until it breaks with a time-out. It cannot be less than two minutes (120,000 ms). The (recommended) default is three minutes (180,000 ms).

Tab Auto save

This tab allows for defining the auto-saving interval, or to dis/enable auto-saving. For further info see Working with EUROMOD - Changing countries' tax-benefit-systems - Saving, saving as and auto-saving, paragraph *Auto-saving*.

Tab Warnings

The user interface issues at appropriate points warnings with the option "Do not reshow this warning". If the respective box is checked this type of warning is not reshown, even if the interface is closed and opened again. The tab *Warnings* allows for defining which types of warnings should be displayed.

Using the help system

To access the user interface's help system press the ***F1 key*** or click the button *Help* in the ribbon *Help & Info*. This leads directly to the help page describing the part of the interface you are working on or, if no specific help is available, to the help table of content.

Moreover, pressing the ***F5*** or ***F6 key*** in the main window of the user interface leads to an explanation of the selected function (if any function is selected), where the *F5* key leads to a descriptive explanation, usually with a couple of examples, whereas the *F6* key leads to a full (but brief) description of the parameters of the function.

Getting info on User Interface Version

The version of the User Interface one is currently working with can be obtained by clicking the button *Version* in the ribbon *Help & Info*.

Administration of countries

Adding new Countries/Add-ons

Adding countries

To add a new country to EUROMOD select the ribbon *Administration Tools* and click the button *Add Country*. This opens a dialog allowing indicating the *Long Name* and the *Short Name* of the country. The former is the countries full name (e.g. Hungary, Germany, Greece, Spain, etc.), the latter is the country code (e.g. HU, DE, EL, ES, etc.).

Moreover, you are asked to indicate a *Flag* in portable network graphic format (png). Ideally the size of the image is 28x17 pixels. You can also select the image with the file search dialog opened by pressing the button alongside the *Flag* field. If you do not indicate a flag image, a default image with a question mark will be used.

Confirming the specifications with *OK* adds the country and instructs you to go to the country gallery (i.e. the ribbon *Countries*) to load the new country. For information on the files, which are created by the add process see EUROMOD Installation and Architecture.

Adding add-ons

You may have noted that the ribbon *Administration Tools* contains a button for deleting add-ons but no button for adding add-ons. This is due to the fact that add-ons are usually not generated from scratch but either exported from a country (see Working with EUROMOD - Importing and exporting add-ons) or created by adapting an existing add-on (see Working with EUROMOD - Saving, saving as and auto-saving).

Deleting countries

To delete one or more countries from EUROMOD select the ribbon *Administration Tools* and click the button *Delete Country*. In the appearing dialog select the countries you want to delete and confirm by pressing the *Delete* button. Note that the action is final and cannot be undone. Also note that the countries must not be loaded - if they are an error message is issued.

For information on the files, which are deleted by the process see EUROMOD Installation and Architecture.

Deleting add-ons

To delete one or more add-ons from EUROMOD select the ribbon *Administration Tools* and click the button *Delete Add-On*. The process is the same as for deleting countries.

Importing countries

To import a country to EUROMOD select the ribbon *Administration Tools* and click the button *Import Country*. This opens a dialog which requests two pieces of information:

Import Country Folder

This folder is supposed to contain the country's XML-files (see EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Format of country parameter files). You can type the *Import Country Folder* or select it with the search dialog, opened by pressing the button alongside the field.

Moreover, the folder may contain the country's flag as an image file named *cc.png*.[1] If the folder does not contain a flag image, a default image with a question mark will be used.

Short Name

If the country does not already exist, the *Short Name* is the name of the *Import Country Folder*.[2] For example, if you are importing Barbados the short name would probably be *BB*. If the country is another version of an existing country, the *Short Name* indicates an alternative short name for this second version. For example, if you are using an alternative version of Belgium you may indicate *B2* as *Short Name*.

Confirming the specifications with *OK* imports the country[3] and instructs you to go to the country gallery (i.e. the ribbon *Countries*) to load the imported country.

[1] *cc* stands for the country's short name (e.g. FR for France, EL for Greece, ...). The flag ought to be in portable network graphic format (png) and is ideally sized 28x17 pixels.

[2] and the country's acronym as used in the names of the country's XML- and flag-files.

[3] That means the user interface copies the folder into the countries' folder. Moreover, if the *Short Name* is different from the name of the *Import Country Folder*, it does all the renaming of files and adapting of parameter files for you.

Exporting & Importing Systems

Exporting systems

To export (a) system(s), select the ribbon *Country Tools* and click the button *Export System(s)*. In the appearing dialog select the system(s) you want to export and indicate an *Export folder*. You can use the button beside the field to open a file search dialog, allowing for selecting a respective path. If you want to remove the exported systems from the country select *Export and delete*, if you want to export the systems without removing them select *Export only*. Click *OK* to start the procedure.

The export procedure creates a folder named after the country's short name (e.g. a folder "DK" if systems are exported from Denmark), which encloses parameter files, which contain the exported systems. In fact the procedure is very similar to the save-as procedure (see Working with EUROMOD - Saving, saving as and auto-saving), with mainly two differences. Firstly (and obviously) the not exported systems are removed from the "export country's" parameter files and, if the option *Export and delete* was selected, the exported systems are removed from the origin country's parameter files. Secondly (and less obviously) the origin country's as well as the export country's parameter files are "cleaned". That means all policies, functions or parameters, which contain only *n/a* values for all remaining systems, are removed. To understand this, imagine having implemented a reform scenario by using many new policies, functions and parameters. If the reform's system is exported (for later use or storage purposes) it would leave a lot of garbage in the country's parameter files. If however all elements only used for the reform are set to *n/a* in all other systems, the export procedure cares for their removal.

Note that the export procedure also stores the data configurations of the exported systems, and, if the option *Export and delete* was selected, deletes them in the country's parameter file.

Importing systems

To import systems to a country select the ribbon *Country Tools* and click the button *Import System(s)*. In the appearing dialog first select the folder where the systems are stored by using the folder button. This could be either a folder created by the export procedure (see above) or a folder created with the *Save As* procedure (see Working with EUROMOD - Saving, saving as and auto-saving) or even another country's folder. Once an appropriate folder is selected, the systems contained in the respective parameter files are displayed in the *Systems* list. Select one or more systems and click *OK* to start the procedure.

The import procedure has to match the imported system(s) with the systems already included in the country's parameter file. That means, in order to present them in the country's spine, it has to add policies, functions or parameters, which only exist in the imported systems. These elements are set to *n/a* in the existing systems, amongst others, to allow for "tidy" removal with the export procedure (see above).

Note that the import procedure asks for a name for an imported system (only) if a system with the same name already exists. Moreover, if the country of the imported system(s) does not correspond with the import country, the procedure asks whether the polices should be renamed respectively, i.e. for example from *ILDef_ee* to *ILDef_hu*, *tin_ee* to *tin_hu*, *bsa_ee* to *bsa_hu*, etc. if an Estonian system is imported to Hungary. This allows aligning the respective policies side by side.

Note that the import procedure also imports the data configurations of the imported systems, if it finds respective information in the import parameter files.

Also note that the import dialog offers a checkbox *Match by Unique Identifier*. The box is only enabled when such an import is possible, for which the first (but not only) condition is, that the systems are imported from another version of the same country. For further details please consult Working with EUROMOD - Comparing versions of a country, paragraph *Importing systems by matching unique identifiers*.

Finally note that the changes caused by importing or exporting systems cannot be undone by using the undo functionality. However, the user interface produces a backup before starting the action, which can be restored via the button *Restore* in the ribbon *Country Tools*. For more information see (Working with EUROMOD - Backup - Restore).

Importing & Exporting add-ons

Importing add-ons

To import an add-on, select the ribbon *Country Tools* and click the button *Import Add-On*. In the appearing dialog first select one of the add-ons listed in the combo-box. Once an add-on is selected, the systems of this add-on are listed in the left box. Select the add-on system you want to import. Once an add-on system is selected, the right box lists all systems of the country which can be used as base for the selected add-on system. Select one of the systems and click OK. This generates the respective add-on system and imports it into the country. For further information on importing systems see Working with EUROMOD - Importing and exporting systems. For further information on add-ons see EUROMOD functions - EUROMOD add-ons and the special functions *AddOn_Applic, AddOn_Pol, AddOn_Func* and *AddOnPar*.

Exporting add-ons

To export an add-on, select the ribbon *Country Tools* and click the button *Export Add-On*. The appearing dialog asks for the following information respectively offers the following options:

- **Long Name:** Indicate a descriptive name for the add-on (e.g. marginal tax rates).
- **Short Name:** Indicate a short name for the add-on, which will be displayed in the add-on gallery. Ideally use two or three-character abbreviations (e.g. MTR). Note that you cannot use an abbreviation, which is already in use for another add-on, nor the short name of a country (i.e. AT, BE, ..., UK).
- **Symbol:** Indicate a picture in portable network graphic format (png). Ideally the size of the image is 32x32 pixels. You can also select the image with the file search dialog opened by pressing the button alongside the *Symbol* field. If you do not indicate a symbol, a default image will be used.
- **Add-On System / Base System:** The

"technique" to generate the add-on is to identify the differences

between the indicated *Add-On System* and *Base System*, in order to

summarise the differences in the add-on. That means, for example, if a policy is only used in the add-on system but not in the base system, a respective *AddOn_Pol* function is generated in the add-on. Note that "not used" in this context is identified by the switch of the base

system being set to *n/a*.

For further information on add-ons see EUROMOD

functions - EUROMOD add-ons and the special functions *AddOn_Applic, AddOn_Pol, AddOn_Func* and *AddOnPar*.

- **Use symbolic identifiers:** If this option is not selected the add-on refers to elements of the base system (policies, functions and parameters) by their GUIDs (**G**lobally

  Unique **Id**entifiers). This is the recommended approach if there are no plans to use the add-on on for several systems or countries. If the option is selected, the GUIDs are exchanged by not system specific symbolic identifiers.

  However, developers need to be aware that symbolic identifiers use an identification mechanism which is more flexible (e.g. it supports using the add-on system for several systems or countries), but also less secure.

  Particularly symbolic identifiers are not unique. For further information see EUROMOD Functions – Identifiers and the placeholders =cc= and =sys=.

- **Use country placeholder:** This option can be selected if the add-on is to be used for several countries. It changes references to policy names from e.g. *yse_sl* to *yse_=cc=*, and taxunit

  parameters from e.g. *individual_sl* to *individual_=cc=*. For further

  information see EUROMOD Functions – Identifiers and the placeholders =cc= and =sys=.

- **Export and delete / Export only:** If the former option is selected, the add-on system is deleted after generating the add-on. Moreover, the parameter file is "cleaned" (see the respective paragraph in Working with EUROMOD - Importing and exporting systems for a description of the cleaning process). If the latter option is used, the country's parameter file is not changed.

Confirming the specifications with *OK* generates the respective add-on and instructs you to go to the add-on gallery (i.e. the ribbon *Add-Ons*)

to view the new add-on.

Note that using the export mechanism for generating an

add-on that can be used for another country than the one it was extracted from must be seen as a starting point only. It is most likely that the developer, after exporting, has to do some adaptions to make the add-on "multi country compliant". As an example, the export mechanism may link an add-on policy, which is located before the standard output policy to the last country (specific) policy (by using *Insert_After_Pol*) instead of the (common) standard output policy (by using *Insert_Before_Pol*).

Also the above mentioned insecurity of using symbolic identifiers (and, though less critical, country placeholders), should be taken into account.

Also note that the changes caused by importing or exporting

add-ons cannot be undone by using the undo functionality. However, the user interface produces a backup before starting the action, which can be restored via the button *Restore* in the ribbon *Country Tools*. For more information see (Working with EUROMOD - Backup - Restore).

Backup - Restore

Some actions, like for example importing and exporting systems, do not allow for undo (see Working with EUROMOD - Undo and redo).[1] However, in order to still enable going back to the state before the action, the user interface generates a backup of the XML-files, which allows for restoring this state.

Backup

Before the action starts, the user interface stores all changes to the XML-files and clears the undo-list. The latter means that no previous actions can be undone anymore. Then it generates the backup in form of a folder, which is stored in the backup-folder of the user interface's temporary folder (see EUROMOD Installation and Architecture - EUROMOD content (parameter files) – Organisation of files). The folder is called *cc_yyyy-mm-dd_hh-mm-ss*, for example *RO_2013-12-29_14-12-00*, for a backup of Romania on the 29th December 2013, at twelve past two pm.[2] After the (successful) termination of the action the user interface issues a message, which indicates the existence of the backup, how the file is called and where it is stored. The message also informs about the possibility to restore this version via the button *Restore* in the ribbon *Country Tools*.

Note that backups are stored for three days in the temporary folder. Older backups are automatically removed by the user interface.

Restore

Clicking the button *Restore* in the ribbon *Country Tools* opens a dialog allowing for selecting the backup-folder as described in paragraph *Backup* above.[3] Once the appropriate folder is selected, the user interface performs the restore and informs about success (or failing).

[1] The concerned "actions" are quite complex procedures, in fact a collection of connected actions.

[2] As one may guess, the folder contains Romania's XML-files (RO.xml and RO_DataConfig.xml). If the backup concerns an add-on, it still takes the form of a folder called e.g. *MTR_2014-01-12_10-57-18*, containing the add-ons XML-file (i.e. MTR.xml).

[3] In fact the backup-folder must contain the country's (or add-on's) XML-files, but actually it is not necessary, that the folder was produced by a backup generated via the user interface.

Comparing versions of a country

It may happen that a country is concurrently adapted by two (or more) developers - for example the country is worked on by an Essex developer as well as the country team - which usually asks for later consolidating of the different versions. The user interface offers support for this task by providing an option to compare versions of a country.[1]

Performing the comparison

Open (the user interface's version of) the country and click the button *Compare Versions* in the ribbon *Country Tools*. This opens a dialog allowing for selecting the country folder to compare with.[2] Once the comparison version is selected the user interface starts its task, whereat it firstly searches for a system with the same unique identifier (in the following abbreviated as UID) in both versions. If it does not find such a system it reports that a comparison is not possible. The user interface uses this "match system" to "symmetrise" the versions, i.e. to take account of the fact that each of the versions may contain components (policies, functions, parameters) which do not exist in the other version, and thus cannot be displayed side by side with a correspondent component (that means, amongst others, that components which do not exist in the interface's version need to be added at an appropriate place).

Further on the user interface imports all systems of the comparison version. The names of the imported systems are prefixed if with *1_* (e.g. system *BE_2010* is called *1_BE_2010*)[3], unless no such system exists in the user interface's version, in which case the name is not changed (e.g. system *BE_2014*, which exists only in the comparison version is still called *BE_2014*). Then the user interface compares each "pair of twin systems", where the twins are identified as systems with the same UID in the user interface's and the comparison version. Please note, that systems with just equal names but unequal UIDs are not recognised as twins! The imported system is still called e.g. *1_BE_2013*, however point c) below applies instead of point b).

The comparison of twin systems consists of discovering differences in the values of policies, functions and parameters of the two systems, where these components are again matched by their UID.

Results of the comparison

There are three possible comparison results for each system imported from the comparison version:

- No differences are found between the twin systems: In this case the user interface removes the imported twin (e.g. system *1_BE_2010*), as it does not provide any additional information. The user is informed about the equality by an information box displayed after the comparison.
- The twin systems show differences: In this case the user interface highlights the concerned cells with colour and expands all policies/functions which show differences. More precisely it implements a "Differences to Base System Formatting" (see Working with EUROMOD - Formatting - Conditional formatting).
- The imported system does not have a twin: In this case the user is informed about this new system by an information box displayed after the comparison.

The information box displayed after the comparison may also report the following: *"For differences which cannot be reflected in the spine move the mouse over the red and green info markers in the group column."*. The next paragraph explains what this means.

Differences covered by "info markers"

If the above mentioned information is issued, the *Group* column shows so called "info markers", which are little red or green squares. Moving the mouse over a cell with an info marker shows an information box, which tells for example:

> refers to system(s): 1_BE_2005 1_BE_2006 1_BE_2007 1_BE_2008 1_BE_2009 1_BE_2010 1_BE_2011 1_BE_2012 1_BE_2013
>
> (info created 27/11/2013 15:33:06)
>
> comment: only excess amount over 11.51 LVL is paid

This info denotes that the concerned parameter, function or policy has a different comment in the comparison country (see 3rd line). The user interface uses this form to hint at different comments in the two version, because comments are not system specific and thus there is no obvious "place" (like for parameter values and switches) where to indicate the difference. The first two lines are shown with each info box and are just info on when the markers were created and which systems are concerned (i.e. the systems imported from the comparison version).

Possible differences indicated by info markers

Differences in comments are not the only and not the most relevant not system specific discrepancy. More relevant are for example differences in the order of policies, functions and parameters, which may be indicated like the following example shows:

refers to system(s): 1_BE_2005 1_BE_2006 1_BE_2007 1_BE_2008 1_BE_2009 1_BE_2010 1_BE_2011 1_BE_2012 1_BE_2013

(info created 27/11/2013 15:33:06)

function order:

1. BenCalc

3. ArithOp

2. Elig

4. BenCalc

5. BenCalc

6. Allocate

This means that in the comparison version the function *Elig* comes before the function *ArithOp* (the listing refers to the order in the comparison version, while numbers refer to the order in the user interface's version). Note that parameter- and function-order differences are indicated by an info marker alongside the function respectively policy containing them, whereas policy-order differences are indicated by an info marker alongside the first policy (as the "containers" of policies, i.e. systems, do not have a *Group* column where the info marker could be displayed).

More relevant not system specific differences are denoted by red info markers. They concern policy/function/parameter order, policy names, parameter settings listed in the *Policy* column (e.g. names of constants/variables in functions *DefConst/DefVar*), private settings of policies/functions and parameter groups.

Less relevant not system specific differences are denoted by green info markers. They concern comments and private comments.

Handling info markers

In order to be able to copy parts of the information provided by the info boxes, e.g. the text of a comment, one can display the information in a little editable window. To do so, select the cell with the info marker and press the *F7* key.

Together with the *Compare Versions* button the user interface provides three further buttons: *Remove Info Markers*, *Store Info Markers* and *Load Info Markers*. To understand this, one needs to know that info markers are not permanent, in the sense that, if the interface is closed and reopened, the info markers are gone.[4] This may be bothersome if for example one wants to continue working only the next day, but still not lose the info markers. Therefore the two latter buttons (*Store*, *Load*) allow storing info markers before closing and reloading them on reopening the interface.[5] The former button (*Remove*) allows for removing info markers to get rid of possibly disturbing red and green squares.

Differences not covered by the comparison

The comparison performed by the *Compare Versions* option only concerns what is displayed in the main view of a country. That means any differences in data settings, system settings and conditional formatting are not covered.

Importing systems by matching unique identifiers

The *Import Systems* dialog (see Working with EUROMOD - Importing and exporting Systems) provides a checkbox *Match by Unique Identifier*. The box is only enabled when such an import is possible. Loosely speaking that means that the systems need to be imported from another version of the same country. More precisely it means that there needs to be at least one system with the same UID in the country from which the systems are imported. If the box is ticked, the import follows the same rules as in the *Compare Versions* procedure. That means it matches the systems by using the UIDs of systems, policies, functions and parameters.

The advantage of this approach is that one may get a better conformity between the existing and the imported systems. The main disadvantage is however, that one may not get runnable systems, as the approach does not correct for not system specific differences. Most importantly it does not correct for different orders of policies and functions, for different settings of parameter groups (*Group* column) and for renamings in the *Policy* column (e.g. changing a component of an incomelist or renaming a constant). Instead it leaves the respective adaptions to the user by providing info markers as described above. (If necessary) The user is warned about the discrepancies and informed about the existence of info markers by a message issued after the import procedure.

"Undoing" the changes caused by the comparison

Note that the changes caused by the comparison cannot be undone by using the undo functionality. However, the user interface produces a backup before starting the comparison, which can be restored via the button *Restore* in the ribbon *Country Tools*. For more information see (Working with EUROMOD - Backup - Restore).

---

[1] For the sake of simplicity the following descriptions refer to countries, however comparing versions is available for add-ons as well.

[2] For information on the country folder see EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Format of country parameter files).

[3] More precisely, the systems are prefixed with consecutive numbers, i.e. if e.g. *1_BE_2010* exists in the user interface's version too, the imported system is called *2_BE_2010*.

[4] Technically that means that they are neither stored in the country XML-files nor in the user settings (like e.g. bookmarks).

[5] The ability to store and reload info markers is the main reason for equipping the information with a date. This hints at the fact that the displayed information may not be up-to-date anymore.

Generating a EUROMOD public version

A EUROMOD public version is a version that is released for public use. Essentially, private countries/add-ons, systems, policies, functions, parameters, datasets and comments are removed from such a version[1], i.e. parts, which are not yet ready or not foreseen to be visible to the public.

User interface support for generating a EUROMOD public version is available by selecting the ribbon *Administration Tools* and pressing the button *Public Version*. Note that a public version can only be generated if all countries are closed. A respective message is issued if any country is open. The appearing dialog allows for specifying a path for the public version, as well as the version number. Note that both specifications are compulsory. Pressing *OK* starts the process[2], which, if terminated successfully, indicates where the public version was stored. Note that the public version is not loaded. To check the result one has to change the user interface's current "content" via the main menu's item *Open project* (see Working with EUROMOD - Open project).

To check which countries/add-ons, systems, policies, functions, parameters and datasets are private before generating the public version (or in fact whenever requested) click the dialog's button *Check Private*. This generates a simple text file listing the private components. The file, called *PrivateComponents.txt*, is (preliminary) stored in the temporary folder (see EUROMOD Installation and Architecture) and opened after its generation.

See Working with EUROMOD – Changing country settings, Working with EUROMOD – Changing system settings, Working with EUROMOD – Setting policies, functions and parameters private, Working with EUROMOD – Configuring datasets and Working with EUROMOD – Private Comments for information on how to set components private.

[1] In addition all user-set colours are removed. Morover default conditional formatting is installed (see Working with EUROMOD - Conditional Formatting - paragraph "Automatically setting the Base System Formatting" and node that any formats, other than the default are overwritten).

[2] The process generates a copy of the folder *EuromodFiles* (see Installation and File Structure - Organisation of files) at the indicated path. This copy is then adapted by removing all private systems, policies, functions and datasets. If all systems of a country are private, the whole country is deleted.

Handling of the decimal and thousand separators in EUROMOD

The user interface uses point as decimal separator and does not allow for thousand separators, e.g. 1234.6 is a valid number and will be interpreted as one thousand - two hundred - thirty four point six. 1.234 is a valid number and will be interpreted as 1 point two hundred - thirty four. 1,234.6 and 1.234,6 are no valid numbers.

Data may use point or comma as decimal separator (see Working with EUROMOD - Configuring datasets).

Output is always produced with the decimal separator configured on the user’s PC (see EUROMOD Basic Concepts - EUROMOD input and output).

Obtaining information upon updating progress

The ribbon *Administration Tools* provides a button *Updating Progress*, which allows for opening a dialog that provides options to obtain information upon the model’s "updating progress". That means EUROMOD developers (and users) can assess which policy years are implemented for each country and which datasets are available to run the respective systems.

The dialog is organised as a register with four tabs. The first tab allows for controlling the information which is to be provided, while the latter three display the respective information:

Tab Systems: displays a table which lists the policy years (columns) which are implemented for each country (rows). The crossing cell indicates whether the respective system is "available", i.e. presumably ready for use, "private", i.e. presumably not yet ready for use or not existing: "-". These policy year columns only display standard systems, i.e. systems following the naming convention *cc_yyyy*. Other available systems, which are not covered by this, are listed in the column *other*. If such a system is private, this is indicated in brackets.

Tab Datasets: provides information on the available datasets for each country. The columns display data years, which are extracted from the dataset name, i.e. if the dataset's name is *cc_yyyy_xxx* (e.g. *be_2007_a3*, *uk_2009_lpc1*, etc.) the dataset is listed in the respective column. There are three additional columns: The column *hypo* lists hypo data, the column *training* lists training data and the column *other* lists other datasets which do not follow the naming convention (i.e. essentially *sl_demo_v4*). The rest of the columns indicate the datasets' properties: i.e. their collection and income year and whether they are private or not.

Tab Combinations: displays for each country the possible combinations of implemented systems and available datasets, where the crossing cells indicate whether this combination is only available (*x*) or a best match (*best*). The information corresponds to that shown in the data configuration dialog (see Working with EUROMOD - Configuring datasets) and just allows getting information on several countries at once.

Tab Settings: As mentioned above, this tab serves controlling the information which is displayed in the other tabs. The list on the left hand side allows for selecting the countries which are included, which buttons to select *All* or *No* country. Clicking the button *Generate* produces respectively refreshes the information as described above for the selected countries. The checkboxes on its left allow for including/excluding parts of the information. The box *Export to text files* provides options for outputting the current content of the three information tabs into text files. The field *Folder* must indicate an output path where to store the resulting text files. The folder can be selected with the button right of the field. The three fields below must indicate the file names for storing the information. The checkboxes left of them allow for including/excluding the respective tab.

Note that the information to generate the tables is taken from the indications as provided in the countries' configuration dialogs (see Working with EUROMOD - Changing system settings and Working with EUROMOD - Configuring datasets) and is as accurate and up-to-date as this base.

Save country XML-parameter-files formatted

The ribbon *Administration Tools* provides a button *Save Formatted*, which allows for saving one or more countries' XML-parameter files either as tab-delimited text files or (still) as XML-files (but) with line breaks. To do so, in the dialog, select the countries which are to be stored formatted. The buttons *Select All* and *Select No* allow for selecting respectively deselecting all countries. Use the respective radio buttons to specify whether you want to store the files in *Text Format* or *With Line Breaks*. Finally specify the folder where the text files should be stored in the field *Export to Folder*. You may use the button right of the field to select a folder. Press *OK* to perform the storage.

Concerning the option *Text Format*, please note that only part of the information stored in the XML parameter files is available in the text files. In principle the text files include what one sees in the expanded policy spine. Information on system settings, private comments, etc. are not available. Moreover, the text files contain only (part of the) information stored in *cc.xml* but not information stored in *cc_DataConfig.xml*. Also note that the first column indicates what a line refers to, i.e. a POLICY[1], FUNCTION or PARAMETER.

For more information on the XML-parameter files see EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Format of country parameter files.

[1] or reference policy: POLICY (REF)

Administration

of EUROMOD variables

EUROMOD

variables in a nutshell

All countries implemented in EUROMOD use the same set of variables to store information taken from input data as well as generated by the model. For example, a variable called *yem* is used by each county to store employment income (taken from input data). As another example, a variable called *bch_s* stores child benefits calculated by the model. This approach does not only care for high comparability over countries but also supports exchangeability in facilitating task like, for example, taking some policy measure from one country and implementing it in another.

At first view the names of EUROMOD variables may seem rather none intuitive (e.g. *dag* for age). In

fact, the names follow an elaborated naming scheme, which is described in detail in the Data Requirement Document (DRD). This brief description confines itself to mentioning some features, which are useful in identifying the meaning of a variable:

- Variables generated by the model end with _s for simulated, whereas variables taken from input data do not have such an ending. For example bch is a

  child benefit taken from data, whereas bch_s

  is a simulated child benefit.
- The first character of a variable's name

  indicates its type, where **b** stands for **benefit** (e.g. bch: b=benefit, ch=child), **t** stands for **tax** (e.g. tin: t=tax, in=income), **p** stands for **pension** (e.g. poa: p=pension, oa=old age), **d** stands for **demographic** (e.g. dag: d=demographic, ag=age), **l** stands for **labour market** (e.g. les: l=labour market, es=economic status), **y** stands for **income** (e.g. yem: y=income, em=employment), **a** stands for **assets** (e.g. afc: a=assets, fc=financial capital), **x** stands for **expenditure** (e.g. xcc: x=expenditure, cc=child care), **k** stands for **in kind** (e.g. ked: k=in kind, ed=education) and **s** stands for **system** (e.g. stm01: s=system, tm=temporary variable).

- The rest of the name is composed of

  two-character acronyms. Take for example the variable *tsceehl_s*: **t** stands for **tax**, **sc** stands for **social insurance contributions**, **ee** stands for **employee**, **hl** stands for **health** and **_s** stands for **simulated**.

  Thus the variable stores employee's health social insurance contributions as calculated by the model.

EUROMOD variables are stored in a special EUROMOD parameter file, the variable description file *VarConfig.xml* (see EUROMOD Installation and Architecture - EUROMOD

content (parameter files) - Organisation of files for the storage place and EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Format of the variables file for the content of this file). The file contains the names of the available variables together with their properties.

Some of the properties serve the model to distinguish if certain routines should be applied on the variable, for example only monetary variables are up-rated. Others are mere information, for example, there is a verbal description of each variable (*Automatic*

Label), and a special description for each country. Moreover, the variable description file stores the acronyms of which the variable names are composed.

Note that in addition to these standardised variables, there are variables which are defined during the model run using the functions *DefVar* and *DefConst* (see EUROMOD

Basic Concepts – EUROMOD terminology paragraph *EUROMOD variables*).

Administrating

variables and acronyms in the EUROMOD user interface

The EUROMOD user interface provides a tool, which allows administrating the information stored in the variable description file. To assess this tool press the button *Variables* in the ribbon *Administration Tools*.

The left part of the tool's surface shows the list of all available EUROMOD variables. For each variable there is a checkbox indicating whether the variable is monetary (checked) or not (not checked) and a verbal description. This description is called *Automatic Label*, indicating that it is automatically generated out of the acronyms building the variable's name, thus the user cannot edit it. Moreover, if a variable is selected, country specific descriptions are listed below the list of variables. These descriptions are editable.

The right part of the tool's surface shows all available acronyms organised in a tree in three levels. The first level exhibits the type as described above (benefit, tax, income, ...). The

second level subdivides types into so called "levels". The levels mainly serve a better overview.[1] Finally, the third level shows the acronyms themselves together with a verbal description, which is used to generate the *Automatic Labels* of the variables using the respective acronym. If an acronym is categorical (e.g. gender has two categories: male and female), selecting the acronym lists the respective categories below the list of acronyms.

Note the possibilities of adapting the size of the window, and therewith the lists, according to your

requirements. In particular note that the window is vertically tiled, i.e. if you move the mouse between the upper (*Variables,*

Acronyms) and lower lists (*Descriptions,*

Categories) a cursor appears, which allows to resize the upper and lower window part.

[1] The level also plays a role in the naming rules of variables. For more information see Working with EUROMOD - Administrating variables.

Administrating Variables

The following operations are performed with the **variable administration tool**. To open this tool press the button *Variables* in the ribbon *Administration Tools* (also see Working with EUROMOD - Administration of EUROMOD variables paragraph *Administrating variables and acronyms in the EUROMOD user interface*).

Adding variables

Click the button *Add Variable* in the ribbon *Variables* of the variable administration tool. Alternatively use the key combination *Alt-V*. This adds an empty row to the list of variables. At first instance the row is added below the selected row. Note, however, that resorting the list (manually or by an automatic update due to another change) moves empty rows to the beginning (ascended sorting) or end (descended sorting) of the list of variables.

Changing the name of a variable

The name of a variable is changed by simply editing the respective cell.

If the name of an existing variable is edited, the acronyms currently used by this variable are highlighted in the list of acronyms. This indicates which acronyms can furthermore be used and which not. In this context, note that only acronyms of the same or a higher level as its precedent acronyms can be appended. See the Data Requirement Document (DRD) for detailed information concerning the generation of variable names.

Once editing is finished, the interface checks the validity of the new name. Firstly, it checks whether the type and acronyms used exist. Secondly, it checks whether the order of acronyms is correct. Finally, it checks whether this name already exists. If any of the checks fails a respective warning is issued, but it is still possible to accomplish the change. The interface automatically updates the variable's description (*Automatic Label*). For unknown types or acronyms question marks are displayed.

Note that changing the name is only possible without warning, if the variable was added during the current session of the administration tool. For existing variables a message warns about possibly renaming a variable used in country implementations.

Changing the monetary state of a variable

To change the state of a variable from non-monetary to monetary or vice-versa, check respectively uncheck the box alongside the variable name.

Changing the country specific descriptions of a variable

If a variable is selected, the *Descriptions* list (located below the *Variables* list) shows country specific descriptions for this variable. These descriptions can be changed by simply editing the respective cells.

Deleting variables

Select the variable to delete and click the button *Delete Variable* in the ribbon *Variables* of the variable administration tool. Alternatively use the key combination *Ctrl-Shift-V*. A message is issued to warn about possibly deleting a variable used in country implementations. This warning is however not shown, if the variable was added in the current session of the administration tool (and therefore cannot be used yet).

Filtering variables

To obtain a better overview the variables listed can be reduced to those having certain properties (monetary / non-monetary, taken from data / generated by the model) and types (DEMOGRAPHIC, LABOUT MARKET, ..., UNKNOWN). Moreover, variables can be restricted to those having a country specific description (for a specific country or for any country). Select the respective properties and/or types and click the button *Apply Filters* to list the variables fulfilling the criteria.

The buttons *Select All Filters* and *Unselect All Filters* serve the easier selection of filters by generating an original state (all filters selected or no filters selected) that then can be refined as required.

Sorting variables

By clicking the header of the *Name* column variables are sorted by name. A second click changes the sorting direction (from ascending to descending and vice versa). Variables can also be sorted by *Automatic Label*, but the result will not differ much from sorting by name.

Searching variables

The *Search* button at the top right of the ribbon *Variables* allows for checking the existence of the variable as specified in the field above the button. Note that it is possible to use the search patterns *?* and *, where *?* stands for one arbitrary character and * for any number of arbitrary characters. The search will select the first visible match of the search pattern. If more than one occurrence is found or if the only occurrence is hidden due to filtering, an info box appears showing all the matches. Hidden matches will appear in dark grey colour. Visible matches will appear in black colour (blue when hovered) and clicking them will select the corresponding variable in the variables table. Also note that a full variable description tooltip appears if you hover the mouse over any of the matches.

Acronym Administration

The following operations are performed with the ***variable administration tool***. To open this tool press the button *Variables* in

the ribbon *Administration Tools* (also

see Working with EUROMOD

- Administration of EUROMOD variables paragraph *Administrating variables and acronyms in the EUROMOD user interface*).

Listing of acronyms

The variable administration tool lists in its right part all

acronyms available for generating variable names (see the Data Requirement Document (DRD) for detailed information concerning the generation of variable names). The acronyms are organised in a tree with three levels. The first level exhibits the variable ***type***, i.e. DEOMOGRAPHIC, TAX, INCOME, etc. The second level subdivides types into ***levels***. The levels mainly serve a better overview.[1] Finally, the third level shows the acronyms themselves together with a verbal description, which is used to generate the *Automatic*

Labels of the variables, which use the respective acronyms. If an acronym is categorical (e.g. gender has two categories: male and female), selecting this acronym in the *Acronyms* list

displays the categories in the *Categories* list (locate below the *Acronyms* list).

Adding types

Click the button *Add*

Type in the ribbon *Acronyms* of

the variable administration tool. Alternatively use the key combination *Alt-T*. This appends a new empty type row at the end of the *Acronyms* list.

Changing types

To change a type's *Description* (DEOMOGRAPHIC, TAX, INCOME, etc.) or *Acronym* (D, T, Y, etc.) simply edit the respective cell. In the case of changing the *Acronym* the interface checks if the *Acronym* is already used by another type and, if so, issues a respective message and prevents the change to avoid ambiguousness. Moreover, the interface checks whether the *Acronym* is used in the current listing of variables. If so, a warning is issued, which lists the variables concerned. The user is still able to accomplish the change, however the description (*Automatic Label*) of the variables will show up question marks to indicate unknown acronyms.

Deleting types

Select the type to delete and click the button *Delete Type* in the ribbon *Acronyms* of the variable administration tool. Alternatively use the key combination *Ctrl-Shift-T*.

The interface checks whether this type is used in the current listing of variables. If so, a warning is issued, which lists the variables concerned. The user is still able to accomplish the removal, however

the description (*Automatic Label*) of

the variables concerned will show up question marks to indicate unknown acronyms.

Adding levels

Click the button *Add*

Level in the ribbon *Acronyms* of

the variable administration tool. Alternatively use the key combination *Alt-L*. This adds a new empty level row.

Note that, other than for types and acronyms, the order of levels is relevant, thus the interface does not simply append the row, but uses the selected row as orientation: if another level row or an acronym row is selected, the new row is inserted below the respective level. If a type row is selected, the level is inserted as the first level of this type.

Changing levels

To change a level's *description*,

simply edit the respective cell. Note that, other than types and acronyms, levels do not possess an *Acronym* (thus the

respective cell is not editable), reflecting that levels are not directly used in generating the *Automatic Label* of

variables.[2]

Deleting levels

Select the level to delete and click the button *Delete Level* in the ribbon *Acronyms* of the variable administration tool. Alternatively use the key combination *Ctrl-Shift-L*.

The interface checks whether acronyms of this level are used in the current listing of variables. If so, a warning is issued, which lists the variables concerned. The user is still able to accomplish the removal,

however the description (*Automatic Label*)

of the variables concerned will show up question marks to indicate unknown acronyms.

Adding acronyms

Select the level, where to add the new acronym (or select

another acronym within the level) and click the button *Add Acronym* in the ribbon *Acronyms* of the variable administration tool. Alternatively use the key combination *Alt-A*. This appends a new empty acronym row at the end of the level.

Changing acronyms

To change an acronym's *Description* (age, gender, etc.) or *Acronym* (AG,

GN, etc.) simply edit the respective cell. In the case of changing the *Acronym* the interface checks if the acronym is already used within this type or is incorrect, i.e. does not consist of two characters. If so, a respective message is issued and the change is prevented to avoid ambiguousness. Moreover, the interface checks whether an acronym is used in the current listing of variables. If so, a warning is issued, which lists the variables concerned. The user is still able to accomplish the change, however the description (*Automatic Label*) of the variables concerned will show up question marks to indicate unknown acronyms.

Deleting acronyms

Select the acronym to delete and click the button *Delete Acronym* in the ribbon *Acronyms* of the variable administration tool. Alternatively use the key combination *Ctrl-Shift-A*.

The interface checks whether the acronym is used in the current listing of variables. If so, a warning is issued, which lists the variables concerned. The user is still able to accomplish the removal, however

the description (*Automatic Label*) of

the variables will show up question marks to indicate unknown acronyms.

Searching acronyms

The *Search* buttons at the top right of the

ribbon *Acronyms* allow for searching

acronyms.

- The button *Acronyms* searches for the acronym as specified in the fields left of the buttons. This is also the default search if you press Enter in the search text field. Select the acronym type (DEOMOGRAPHIC, TAX, INCOME, etc.) in the upper field and the searched acronym in the field below. Pressing the button selects the matched acronym.

  Note that, if ALL ACRONYMS is selected instead of a specific type, several matches may be found, as acronyms need to be unique only within a type. If more than one occurrence is found, an info box appears showing all the matches. Matches will appear in black colour (blue when hovered) and clicking them will select the corresponding acronym in the acronym tree.

Also note that a full acronym description tooltip appears if you hover the mouse over any of the matches.

- The button *Description* searches for acronyms with a description as specified in the fields left of the button. Select the acronym type (DEOMOGRAPHIC, TAX, INCOME, etc.) in the upper field and the searched description in the field below. Note that search patterns *?* and * can be used (and most likely will), where *?* stands for one arbitrary character and * for any number of arbitrary characters (examples: *child*, *wom?n*). Pressing the button selects the matched acronym.

  If more than one occurrence is found, an info box appears showing all the matches.

  Matches will appear in black colour (blue when hovered) and clicking them will select the corresponding acronym in the acronym tree. Also note that a full acronym description tooltip appears if you hover the mouse over any of the matches.

[1] The level also plays a role in the naming rules of

variables. For more information see Working with EUROMOD - Administrating variables.

[2] They are used indirectly by determining the order of acronyms. For more

information see Working with EUROMOD - Administrating variables.

Saving variables and acronyms

In the variables administration tool's menu select the menu item *Save*, alternatively type *Ctrl-S*. The interface checks for empty variable rows as well as empty or incomplete acronym rows. If it detects such rows a respective message is issued and the user is asked for correction.

To open the variables administration tool press the button *Variables* in the ribbon *Administration Tools* (also see Working with EUROMOD - Administration of EUROMOD variables paragraph *Administrating variables and acronyms in the EUROMOD user interface*).

Importing variables

The following operations are performed with the **variable administration tool**. To open this tool press the button *Variables* in

the ribbon *Administration Tools* (also

see Working with EUROMOD

- Administration of EUROMOD variables paragraph *Administrating variables and acronyms in the EUROMOD user interface*).

To import variables from an external variables definition file (VarConfig.xml) select the item *Import*

Variables from the variables administration tool's menu. This firstly allows the selection of the external variables definition file. Once the file is chosen, a dialog shows the possible changes, i.e. differences between the user interface's internal variables definition file and the external one. This means, the dialog lists the variables and acronyms either existing only in the internal respectively external variables definition file or have different attributes in the two files.

Note that importing variables is only possible if no changes were accomplished in the current session of the variables administration tool, otherwise the menu item is disabled. In this case close and open the tool again, saving the changes if required.

Also note

that, to begin with, it is assumed that the user wants to overtake all modifications from the external variables definition file. Respective changes are suggested in the *Action* columns.

This behaviour can be changed by ticking/unticking

the checkboxes in the *Perform* columns. This is supported by three buttons, which aim to make selecting more efficient.

- Clicking the button *Tick selected variables* ticks the checkboxes of all selected rows in the variables list. Selected rows are characterised by blue background colour. Selecting a single row is accomplished by clicking it. Selecting multiple rows is accomplished by holding the *Strg* key and clicking the respective rows. Selecting a range of rows is accomplished by clicking the first row pressing and holding the *Shift* key and selecting the last row.

- Clicking the button *Untick selected variables* unticks the checkboxes of all selected rows. See above for recognising and choosing selected rows.
- Clicking the button *Add only* unticks the checkboxes of all variables and acronyms for which the *Action* column is set to *delete*.

List of possible changes in variables

The left part of the

dialog shows which variables are different in the external and the internal variables definition file. There are following types of differences:

- A variable exists in the external variables

  definition file only: The column *Action* suggests adding this variable.
- A variable exists in the internal variables

  definition file only: The column *Action* suggests deleting this variable.
- The monetary status of a variable is different:

  The column *Action* suggests changing

  the monetary status and the column *Info* informs about the direction of the change, i.e. to overtake the status defined in the external variables definition file.
- One or more of the country specific descriptions

  of a variable are different: The column *Action* suggests changing the variable and the column *Info* tells that the difference is in different descriptions. The concrete differences are displayed in the *Descriptions* list below the *Variables* list.
  Note that, one can either overtake all descriptions of this variable from the external file (checkbox *Perform* ticked), or keep all descriptions of the internal file (checkbox *Perform* not ticked), partly overtaking is not foreseen.
  Also note, that it is possible (though not recommended) to import a variable definitions file that refers to a different set of countries (e.g. a country was added in the external version, which is not yet implemented in the internal

version). In this case the importing tool issues a warning, telling that it cannot overtake country specific descriptions.

List of possible

changes in acronyms

The right part of the dialog shows which acronyms

are different in the external and the internal variables definition file. There are following types of differences:

- A whole type of acronyms exists in the external

  variables definition file only: The column *Action* suggests adding this type (including all levels and acronyms contained).
- A whole type of acronyms exists in the internal

  variables definition file only: The column *Action* suggests deleting this type (including all levels and acronyms contained).
- The description of an acronym type is different

  (e.g. IN KIND changed to BENEFIT IN KIND): The column *Action* suggests changing the description and the column *Info* tells the new description. Note that a change of the shortcut of an acronym type (e.g. K for IN KIND) is treated as deleting the whole type and adding a new type with the new shortcut, as in fact all variable names using the acronyms of this type get invalid.
- A whole level of acronyms exists in the external

  variables definition file only: The column *Action* suggests adding this level (including all acronyms contained). The tool tries to add the level at the same position as in the external file, by searching for a common predecessor level.
- A whole level of acronyms exists in the internal

  variables definition file only: The column *Action* suggests deleting this level (including all acronyms contained).
- An acronym exists in the external variables

  definition file only: The column *Action* suggests adding this acronym.

- An acronym exists in the internal variables

  definition file only: The column *Action* suggests deleting this acronym.
- The description of an acronym is different: The

  column *Action* suggests changing the

  description and the column *Info* tells

  the new description. Note that, similar to types, a change of the acronym itself is treated as deleting the acronym and adding a new acronym, as in fact all variable names using the acronym get invalid.
- One or more of the categories of an acronym are

  different: The column *Action* suggests

  changing the acronym and the column *Info* tells that the difference is in different categories. The concrete differences are displayed in the *Categories* list

  below the *Acronyms* list.
  Note that, one can either overtake all categories of this acronym from the external file (checkbox *Perform* ticked), or keep all categories of the internal file (checkbox *Perform* not ticked), partly overtaking is not foreseen.

Performing the import

To accomplish the import, tick the checkboxes in the *Perform* columns as required, and press the button *Import*. For

variables you can tick or untick several *Perform* boxes at once by first selecting the respective rows and then clicking the buttons *Tick selected variables* respectively *Untick selected variables*.

Note that it is not possible to undo the import via the

undo-functionality. However, there is no automatic saving of the variables definitions file. That means closing the variables administration tool without saving can still restore the old state.

Cleaning variables

To find out which EUROMOD variables and acronyms are not used and therefore could potentially be deleted select the item *Clean Variables* from the variables administration tool's menu (to open the variables administration tool press the button *Variables* in the ribbon *Administration Tools* - also see Working with EUROMOD - Administration of EUROMOD variables).

Note that cleaning variables is only possible, if no changes were accomplished in the current session of the variables administration tool, otherwise the menu item is disabled. In this case close and open the tool again, saving the changes if required.

Loading not used variables and acronyms**:** Pressing the button *Load* starts the search for unused variables and acronyms. As this may take a view minutes a progress bar is displayed.

List of not used variables**:** The left part of the dialog shows the variables not used in any country's implementation. Initially it is assumed that all listed variables are to be removed. Uncheck any variables which should not be deleted.

List of not used acronyms**:** The right part of the dialog shows the acronyms not used in any variables' name. Note that acronyms used in a variable foreseen for removal (i.e. checkbox ticked) may be listed as unused, if no other variable uses this acronym. Again initially it is assumed that all listed acronyms are to be removed. Uncheck any acronyms which should not be deleted.

Performing the cleaning process**:** To accomplish the cleaning press the button *Clean*. Note that it is not possible to undo the cleaning via the undo-functionality. However, there is no automatic saving of the variables definitions file. That means closing the variables administration tool without saving can still restore the old state.

Applications

The user interface provides access to a couple of external

tools implemented in MS-Excel. Currently available are a tool for generating hypothetical EUROMOD input data and a tool using EUROMOD's output for drawing budget constraint graphs. To access the tools, select the ribbon *Applications* and press the respective button.

Policy Effects tool

What is the Policy Effects tool?

The Policy Effects tool estimates the first-order effects of policies on household incomes. The tool is meant to assist with model validation (see EUROMOD Country Reports on the EUROMOD website) and compares policies in place in two consecutive years, both in nominal and real terms.

Where can I find this?

You can find the Policy Effects tool under the "Applications" toolbar ribbon, inside the "Tools" group.

How does it work?

When you open the Policy Effects tool you are presented with an input form, which allows you to specify the (start) period of interest, countries, input datasets and indexation factors (*alphas*, see Methodology below). By default, the tool is set to analyse policy changes in the most recent policy year available in the model, using the dataset offering the best match for the start of this period (see EUROMOD Basic Concepts - Terminology).

The field *Output path* at the bottom of the form allows changing the default output folder where the tool stores micro-level output for relevant policy scenarios together with log files (standard EUROMOD run logs and a tool-specific log). Note that the output folder must exist, otherwise EUROMOD issues an error message, and any existing files with the same names will be overwritten without a warning.

To produce micro-output and summary results, click on the "Run & Show Results" button.

Results

Once specified policy systems have been simulated and analysed, you will be presented with the results form. This has a tab control (on the bottom) with a separate tab for each country previously selected. Each country tab page has one inner tab control (on the top), with a separate tab for each selected indexation factor (*alpha*).

Each summary table shows the effect of policy changes in a given period on mean equivalised household disposable income by (standard) income component and income decile group, as a percentage of mean equivalised household disposable income in the starting year. The latter is also used to construct income decile groups. Throughout, the modified OECD equivalence scale is used. For example, a 3 percentage point change for public pensions in the third decile group in country A, indicates that changes (increases) in public pensions accounted for a 3 percentage point increase in average equivalised household income. In the case of taxes and SIC, a positive (negative) number implies that taxes/SIC have been reduced (increased). The sum of effects by income component (columns 2 to 8) equals the total (column 9).

You can copy-paste any selected cells from these tables, either by pressing Ctrl+C or by right-clicking and selecting "Copy" from the context menu. You can also export all the calculated statistics (all tables for all countries) into a single Excel file, by clicking on the "Export" button.

Methodology

The estimation of policy changes draws on the method suggested by Bargain and Callan (2010, The Journal of Economic Inequality). Consider a single household and denote its market income (and other characteristics) with *y* and monetary values of tax-benefit parameters as *p*. A function *d(p,y)* calculates household disposable income on the basis of its market income and monetary parameters, reflecting the structure of the tax-benefit system (e.g. tax rates, benefit eligibility rules). In period *t*, household disposable income can be denoted as $d_t(p_t,y_t)$.

The tool estimates the direct effect of policy changes on household incomes in the period from *t=1* to *t=2*. To isolate it from other changes in the income distribution (e.g. changes in household composition or market incomes), household disposable incomes under the two policy systems are assessed holding household characteristics and market incomes constant. Furthermore, to adjust for changes in nominal income levels over time, the monetary parameters of the tax-benefit system are adjusted with a factor *alpha* which reflects benchmark indexation. Specifically, the tool estimates the policy effect (for each household) as:

$$\Delta = d_2\left(\frac{1}{\alpha}p_2, y_1\right) - d_1(p_1, y_1)$$

Note that this is a particular variation of decomposition chosen for EUROMOD validation purposes. (The full decomposition framework is described in BC2010.) Technically, instead of scaling monetary policy parameters, the tool scales monetary input variables with the factor *alpha* and monetary output variables with the factor *1/alpha:*

$$\Delta = \frac{1}{\alpha} d_2(p_2, \alpha y_1) - d_1(p_1, y_1)$$

This relies on the assumption that tax-benefit systems are linearly homogenous, that is *d(cp,cy)=cd(p,y)*. Input variable adjustments are limited to market incomes, expenditures and assets (specifically: all monetary variables which name starts with *y** or *x**, variable *afc* as well as all variables included in *ils_origy*, while excluding variables included in *ils_ben*). Output variable adjustments cover all monetary variables.

Summary results in the table are shown as a percentage of (mean) disposable incomes in the starting year, i.e. $d_1(p_1,y_1)$.

There are two pre-defined choices for benchmark indexation: (i) factor *alpha* could be set to 1 in which case the effect of policy changes is calculated simply in nominal terms, or (ii) CPI indexation in which case the effect of policy changes is calculated in real terms. For (ii), the adjustment factor is automatically derived on the basis of Eurostat's Harmonized Index of Consumer Prices – this information needs to be defined (as *$HICP*) along with other series of uprating indices (in the index table).

Note that the tool cannot be used when uprating factors are not defined in an index table (see Defining Uprating Factors).

EUROMOD

PLUGINS

Plug-ins are software components that extend the functionality of EUROMOD. While the core user interface concentrates on supporting the implementation, adaptation and running of countries' tax-benefit systems, plug-ins provide additional features like producing basic summary statistics, performing microvalidation, or generating hypothetical datasets. In the official release of v1.10 the only publicly available plug-in is the "Summary Statistics", but many more will be added in future releases to accommodate task-specific needs.

Summary Statistics plug-in

What is the Summary Statistics plug-in?

The Summary Statistics plugin is an analysis tool that produces a fixed set of statistics on income distribution based on (a) EUROMOD output file(s).

Where can I find this?

You can find the Summary Statistics plugin under the "Applications" toolbar ribbon, inside the "EUROMOD plugins" group.

How does it work?

When you run the Summary Statistics plugin you are presented with an input form which allows you to specify a path to a base directory. This should be the directory where your output files are stored. You should then specify on which files you want to perform calculations by adding/removing files. Once you have selected the files you want to analyse, click on the "Calculate Statistics" button.

After the files have been read and analysed, you will be presented with the results form. This has an outer tab control with one tab page for each system/file you analysed. Each tab page has one inner tab control, whose tab pages hold different tables with the summarized statistics results. You can copy-paste any selected cells from these tables, either by pressing Ctrl+C on your keyboard or by right-clicking your mouse and selecting the "Copy" command from the context menu. You can also export all the calculated statistics (all tables of all systems/countries) into one single Excel file, by clicking on the "Export" button.

Keyboard Shortcuts

As it happens with most software applications, EUROMOD also understands a number of keyboard shortcuts

that, once you get used to, will help you increase your productivity. Following is a list of keyboard

shortcuts that are generally available, both in the Country spine and in other forms (e.g. Uprating Indices).

Keyboard Shortcut	Usage
Ctrl + Z	Undo the latest action. EUROMOD can store up to 100 actions, after which, the oldest action is removed to make space for the new action. There are also specific actions that reset the undo/redo functionality. These will issue a warning before they are applied and the user can chose to abort them.
Ctrl + Y	Redo the latest undone action.
Shift + Arrows	Starting from the current cell, it will select an area of cells that can be copied.
Ctrl + C, Ctrl + Ins	Copy the selected items into the clipboard. This works for text within cells, but also for whole cells both in the spine and helping forms. The user may select multiple cells / parameters / functions / policies, by holding the Shift button and selecting the start & end location with the mouse.
Ctrl + X, Shift + Del	Cut the selected text or cells and copy them into the clipboard. Note that this does not work for parameters / functions / policies. It only works for selected text within a cell and for tables within helping forms (such as Uprating Indices). Multiple selection works as for Copy.
Ctrl + V, Shift + Ins	Paste the copied items into the current position. Note that this will not create new rows in the spine. So if you need to copy-paste several parameters of one function into another, please make sure that it already has the same number of parameters and in the same order.
F1	Brings up the help. If the focus is on helping form, the help will try to automatically load the corresponding page. For function-specific help, see at the end of the following table.

Following is a list of keyboard shortcuts that are specific to the country spine.

Keyboard Shortcut	Usage
Ctrl + F	This will bring up the "Search" form.
Ctrl + H	This will bring up the "Search and Replace" form.
Ctrl + S	Save the current state of the country you are working on.
Ctrl + Right arrow, "+"	Expand the current function or policy. Note that if you are on a parameter cell, pressing "+" will start editing the cell and add the "+" character (Ctrl + Right arrow will do nothing in this case, as

	there is nothing to expand).
Ctrl + Left arrow, "-"	Collapse the current function or policy. Note that if you are on a parameter cell, pressing "-" will start editing the cell and add the "-" character. Pressing Ctrl + Left while on a parameter or an already collapsed function will move the focus to the parent function/policy.
/	Collapse the current item and all sub-items. On a function works like "-", but on a policy it will collapse both the policy and all functions within it.
*	Expand the current item and all sub-items. On a function works like "+", but on a policy it will expand both the policy and all functions within it.
Ctrl + Up arrow, Ctrl + Down arrow	Move the current parameter / function / policy up or down respectively. Note that moving parameters cannot move outside their parent function and similarly moving functions cannot be moved outside their parent policy. Also works with moving an area selection. Another way to move parameters / functions / policies up & down is to click & drag them with the mouse.
Alt + O	Hide all rows but the selected one. This will hide only rows of the selected level, so pressing it when the focus is on a parameter will hide all other parameters of that function only. You can find more options to hide & unhide rows, by right-clicking on the row numbers.
Alt + S	Spread the value of the current cell on all other systems. The focus must be on a system column.
Ctrl + A	Open the "Add Parameters" box in order to add more parameters into the current function. The focus must be on a function or parameter.
F5	Brings up the help and automatically loads the "Description" help page for the focused function.
F6	Brings up the help and automatically loads the "Summary of parameters" help page for the focused function.

EUROMOD Functions

What are functions and how are they used

EUROMOD functions are building blocks that allow EUROMOD modellers to implement a country's tax-benefit system. Each policy (tax/benefit) is described by one or more such functions. Modellers specify how EUROMOD calculates the policy by setting the parameters of the functions to appropriate values.

What are functions?

The example below illustrates how EUROMOD functions are used to implement a simple child benefit of 100 Euro monthly, received by families with at least one child aged younger than three. The benefit is implemented by using two functions: one describing the eligibility rule, i.e. there must be a child younger than three in the family, and the other describing the calculation of the benefit, i.e. payment of 100 Euro monthly.

Example 1:

Policy	**SL_demo**	*Comment*
Elig	**on**	**eligibility rule**
elig_cond	{dag < 3}	there must be
TAX_UNIT	tu_sben_family_sl	at least one child aged younger than three
ArithOp	**on**	**benefit calculation**
who_must_be_elig	one	
formula	100#m	the benefit amounts to 100 € per month
output_var	bch_s	
TAX_UNIT	tu_sben_family_sl	

Each function consists of a header

displaying the **name of the function**.

In the example the eligibility rule is implemented by a function called Elig and the calculation of the benefit is implemented by a function called ArithOp. Moreover, each function has a "**switch**", defining whether the function is activated or not. In the example both functions are switched on.

The possibility of switching off a function may, for example, be used if a reform scenario is implemented, where there is no eligibility condition, as illustrated in the next example. In this example this results in each family receiving the benefit, irrespective of having a child aged younger than three (or any other condition).

Example 2:

Policy	**SL_demo**	**SL_reform**	*Comment*
Elig	**on**	**off**	**eligibility rule**
elig_cond	{dag < 3}	{dag < 3}	there must be
TAX_UNIT	tu_sben_family_sl	tu_sben_family_sl	at least one child aged younger than three
ArithOp	**on**	**on**	**benefit calculation**

who_must_be_elig	one	n/a	
formula	100#m	50#m	the benefit amounts to 100/50 € per month
output_var	bch_s	bch_s	
TAX_UNIT	tu_sben_family_sl	tu_sben_family_sl	

In addition to name

and switch a function consists of parameters to

specify its behaviour. The leftmost column in the examples contains the **names of the parameters** used, whereas the "SL_demo" headed column contains the **parameter values** for the system "SL_demo". In the second example there is another column for the system "SL_reform", illustrating that parameter values can be different for each system. In the example the amount of the benefit is changed from 100 to 50 Euro in the reform scenario.

Each function offers a different

set of parameters to specify its behaviour. For example, the general purpose of the function Elig is to

implement conditions under which a benefit is received / a tax must be paid.

Therefore, it offers parameters that allow the specification of such conditions. For the child benefit the parameter elig_cond is used. This parameter takes "formulas" with a yes/no result as values, in the example {dag<3}, i.e. the variable for age must be smaller than three. The general purpose of the other function used, ArithOp, is to implement arithmetical operations.

Consequently, it offers parameters to specify the operation. For the child benefit the parameter formula is used to define a

very simple operation – it is just set to the amount of 100 respectively 50 (#m stands for monthly).

Each function, which calculates some result, offers the parameter output_var to

define a variable, which takes this result. In the example the parameter output_var of the function ArithOp is set to the variable bch_s (b=benefit, ch=child, _s=simulated), i.e. this variable takes the amount of the benefit, for families fulfilling the eligibility condition. The parameter output_var is

not used with the function Elig

in the example, thought the function calculates a

result, which is either one, if eligibility conditions are fulfilled, or zero, if not. So where does the function write its result to? The simple answer is that the function has a default output variable called sel_s (s=system, el=eligibility, _s=simulated), which means that, if no other output variable is indicated by using the parameter output_var, the result is written to sel_s.

The next question that may be posed is: how does the function ArithOp know that

it should fill the variable bch_s

with the benefit amount only for those families fulfilling the eligibility condition calculated by the function Elig?

This is accomplished by the parameter who_must_be_elig and explained in section EUROMOD

Functions - Interactions between functions.

Interactions between functions

Usually more than one function is used to calculate a benefit or tax. That means that the functions interact in some way. One could classify these interactions in four categories:

- Condition: one function (usually Elig) evaluates a condition and a subsequent function operates on the basis of the result of this evaluation.
- Input: one function calculates some result,

  which is used as an input by a subsequent function.
- Addition: one function calculates a part of a policy and a subsequent function calculates another part of the policy and therefore needs to add to the first part.
- Replacement (actually not a real interaction): a subsequent function replaces the result of a precedent function, which of course only makes sense if the result of the first function is used in between.

The following examples illustrate these different forms of interaction.

Example 1:

Policy	**SL_demo**	*Comment*
Elig	**on**	**eligibility rule**
elig_cond	{dag < 3}	there must be at least one child aged <3
TAX_UNIT	tu_sben_family_sl	
ArithOp	**on**	**benefit calculation**
who_must_be_elig	one_member	
formula	100#m	the benefit amounts to 100 € per month
output_var	bch_s	
TAX_UNIT	tu_sben_family_sl	

In example 1 the second function, ArithOp, uses the result of the first function, Elig, to

determine whether the benefit is calculated for a family or not, dependent on the value of the parameter who_must_be_elig.

What happens in detail is that the function Elig sets the variable sel_s to one for all persons younger than three and to zero for all older persons. The function

ArithOp uses this information together with the value of the parameter who_must_be_elig.

A value of one_member means that at least one member of the family needs to be eligible, i.e. for at least one person in the family the variable sel_s must be set to one. If this is the case the benefit is calculated for the family, i.e. the variable bch_s set to 100, otherwise no calculation takes place, i.e. the variable bch_s keeps its value of zero.

(To be precise, in the concrete example any possibly existing value of the variable would be overwritten by zero. This is however an alterable behaviour.)

Example 2:

Policy	**SL_demo**	*Comment*
ArithOp	**on**	**calculate lower limit**
formula	xcc * 0.1	lower limit is 10% of child care costs
output_var	sin01_s	write lower limit to intermediate variable
TAX_UNIT	tu_individual_sl	
ArithOp	**on**	**benefit calculation**
formula	100#m	the benefit amounts to 100 € per month
lowlim	sin01_s	with a lower limit as calculated above
output_var	bcc_s	
TAX_UNIT	tu_individual_sl	

Example 2 shows a child care benefit,

which amounts to 100 Euro monthly, with a lower limit of 10% of child care costs. The first function calculates the lower limit, i.e. 10% of the variable for child care costs, xcc

(x=expenditure, cc=child care), and writes it to the intermediate variable sin01_s (s=system, in=intermediate, _s=simulated). The second function uses the result of the first function by setting the parameter lowlim to sin01_s, i.e. applying the value of this variable as the lower limit of its own result, which is written to the variable bcc_s

(b=benefit, cc=child care, _s=simulated).

Example 3:

Policy	**SL_demo**	*Comment*

ArithOp	**on**	**calculate health insurance contributions**
formula	yem*0.05	calculate health contributions as 5% of employment income
output_var	tscee_s	write health contributions to output variable
TAX_UNIT	tu_individual_sl	
ArithOp	**on**	**calculate unemployment insurance contributions**
formula	yem*0.03	calculate unemployment contributions as 3% of employment income
output_add_var	tscee_s	add unemployment contributions to output variable
TAX_UNIT	tu_individual_sl/td>	

Example 3 calculates employee

social insurance contributions as the sum of health contributions (5% of employment income yem) and

unemployment contributions (3% of employment income). The first function calculates health contributions and **writes** its result to the output variable tscee_s

(t=tax, sc=social contribution, ee=employee,

_s=simulated). The second function calculates unemployment contributions and **adds** its result to the output variable tscee_s. Whether the result of a function overwrites the output variable or adds to any previous value is determined by using either the parameter output_var (output variable is overwritten) or output_add_var (result is added to output variable).

Example 4:

Policy	**SL_demo**	*Comment*
ArithOp	**on**	**calculate lower limit for child care benefit**
formula	xcc * 0.1	lower limit is 10% of child care costs
output_var	sin01_s	write lower limit to intermediate variable
TAX_UNIT	tu_individual_sl	
ArithOp	**on**	**benefit calculation child care benefit**
formula	100#m	the benefit amounts to 100 € per month
lowlim	sin01_s	with a lower limit as calculated above
output_var	bcc_s	
TAX_UNIT	tu_individual_sl	
ArithOp	**on**	**calculate lower limit education benefit**
formula	xed * 0.1	lower limit is 10% of education costs
output_var	sin01_s	write lower limit to intermediate variable
TAX_UNIT	tu_individual_sl	replacing previous value
ArithOp	**on**	**benefit calculation education benefit**

formula	100#m	the benefit amounts to 100 € per month
lowlim	sin01_s	with a lower limit as calculated above
output_var	bched_s	
TAX_UNIT	tu_individual_sl	

Example 4 extends the child care benefit of Example 2 by an (similarly designed) education benefit. For both benefits the intermediate variable sin01_s is used for calculating the lower limit. Once the childcare benefit is calculated, its lower limit is no longer used and can be replaced by the lower limit of the education benefit.

Sorts of functions and brief description of available functions

One could classify EUROMOD functions into three categories:

policy functions, system functions and special functions.

Policy functions

These functions are primarily

designed for implementing policy instruments (taxes and benefits). The category contains the following functions:

- **Elig** is referred to as eligibility function as it is most frequently used for determining the eligibility for receiving benefits. However, it also allows for determining the liability for paying taxes, as well as evaluating other conditions.
- **BenCalc** is referred to as the benefit calculator, as it allows for modelling a wide range of policy instruments, in particular benefits.
- **ArithOp** is a simple calculator, allowing for the most common arithmetical operations.
- **SchedCalc** allows for the implementation of the most common (tax) schedules.
- **Allocate** allows for (re)allocating amounts (incomes, benefits, taxes) between members of assessment units.
- **Min and Max** are simple minimum and maximum calculators.

System functions

These functions are designed for

implementing the "framework" of the tax-benefit model. A country’s parameter file contains several special policies to implement this framework.

The section EUROMOD Basic Concepts -

Presentation of countries' tax-benefit-systems provides more information about these special EUROMOD policies and their purpose. Usually, the implementation of a special policy uses just one or two particular system functions. The category contains the following functions:

- **Uprate** is used in the special policy Uprate_cc and allows

  for the uprating of monetary dataset variables.
- **SetDefault** allows for the setting of default values for not existent dataset variables.
- **DefIL** is mainly used in the special policy ILDef_cc and

  allows for the definition of incomelists (see EUROMOD

  Basic Concepts - EUROMOD terminology).
- **DefTU** is mainly used in the special policy TUDef_cc and

  allows for the definition of assessment units (see EUROMOD Basic Concepts - EUROMOD terminology).
- **UpdateTU** allows

  for the redefinition of assessment units.
- **DefOutput** is used in the special policies output_xx_cc and

  allows for the definition of model output (see EUROMOD

  Basic Concepts - EUROMOD input and output).
- **DefVar** and **DefConst** allow for the

  definition of intermediate variables respectively constants, which can be used by other (mostly policy) functions.

Special functions

These functions go beyond the

implementation of the tax-benefit system. The category contains the following functions:

- **Loop** and **UnitLoop** allow for repeating

  part (or all) of the tax-benefit calculations. As an example, for calculating marginal tax rates at least part of the policies need to be calculated twice, once for original income and once for marginally increased income.

- **Store** and **Restore** are mainly, though not

  exclusively, designed to be used with the loop functions described above, as it is frequently necessary to set some or all variables back to their initial (or some other previous) value after each iteration.
- **ChangeParam** allows for changing the value parameters. It also can, for example, be used for more transparent implementation of policy reforms, by avoiding direct changes of the base policy parameters and putting the alterations in separate reform policies instead.
- **Totals** allows for the calculation of aggregates (i.e. sums, means, etc.) of variables or incomelists over the whole population (represented by the dataset) or a selected subgroup.
- **DropUnit** and **KeepUnit** allow for dropping

  (keeping only) individuals, families or households with special characteristics from (within) the calculations.
- **ILVarOp** allows for operations on the content, i.e. the variables of an incomelist.
- **RandSeed** sets the starting point for generating a series of pseudorandom numbers.
- **CallProgramme** allows for calling an external application.
- **DefInput** allows for reading values for one or more EUROMOD variables from a text file.

Common parameters and general features in interpreting their values

All functions offer parameters to determine their behaviour. There are several features with respect to parameters and interpreting their values, which are shared by all or many functions. Moreover, there is a set of **common parameters**, which are provided by all or most of the policy functions and some of the system and special functions.

Moreover, parameters can be categorised into **compulsory** and **optional parameters**. If a modeller tries to use a function without a compulsory parameter EUROMOD issues an error message. If an optional parameter is not indicated a default value is used. The descriptions of the functions list the compulsory and optional parameters and the default values of the latter.

Common Parameters

All or most of the policy functions and some of the system

and special functions provide common parameters. They can be classified into four categories:

Common

parameters affecting output

The parameter **output_var** allows for the indication of a variable for storing the result of the function's calculations. All policy functions provide this parameter. In general the parameter is compulsory, i.e. the modeller must explicitly name a variable that stores the function's result. There is just one exception: The function Elig has the

variable sel_s (s=system,

el=elig, _s=simulated) as its default output

variable. System functions in general have no **output_var** parameter. Some but not all of the special functions provide an **output_var** parameter – the descriptions of the functions in section EUROMOD Functions - Summary of functions and their parameters indicate whether this is the case or not.

The parameter **output_add_var** has the same functionality as the parameter output_var but, in contrast to output_var, where any existing value of the output variable is overwritten, with output_add_var

the function result is added to any existing value of the output variable. The parameter is not compulsory itself, but either output_var or output_add_var must be indicated. The parameter is provided by all functions, which provide the parameter **output_var**, except for the function Elig (as it is not very meaningful to add to a yes/no variable).

The parameter **result_var** allows the indication of a "second output variable". In general this makes only sense in combination with the parameter output_add_var. Example 1, which is an extension of example 3 in section EUROMOD

Functions - Interactions between functions, demonstrates the use of this parameter.

Example 1:

Policy	**SL_demo**	*Comment*
ArithOp	**on**	**calculate health insurance contributions**
formula	yem*0.05	calculate health contributions as 5% of employment income
result_var	tsceehl_s	write health contributions to variable for health sic
output_var	tscee_s	write health contributions to general sic variable
TAX_UNIT	tu_individual_sl	
ArithOp	**on**	**calculate unemployment insurance contributions**
formula	yem*0.03	calculate unemployment contributions as 3% of employment income
result_var	tsceeui_s	write health contributions to variable for unemployment sic
output_add_var	tscee_s	add unemployment contributions to general sic variable
TAX_UNIT	tu_individual_sl	

The two functions in example 1

calculate employee social insurance contributions by adding health and unemployment contributions into the variable tscee_s (t=tax, sc=social contribution, ee=employee, _s=simulated). While in

example 3 of section EUROMOD

Functions - Interactions between functions the amount of the single contributions was lost, in this example the parameter result_var is used to indicate variables which store them: health contributions are stored in the variable tsceehl_s (hl=health insurance) and unemployment contributions are stored in the variable tsceeui_s (ui=unemployment insurance). In general that means that the variable indicated with the parameter result_var takes

the function’s result. Accordingly, any possible existing value of the variable is always overwritten. This optional parameter is provided by all functions, which provide the parameter **output_var**.

Common

parameters affecting "eligibility"

Example 1 in section EUROMOD Functions - Interactions between functions provides an explanation for the parameter **who_must_be_elig**. This explanation however does not mention which values (except from one_member) the parameter can take on. These are:

- one_member (or one): one member of the assessment unit must be eligible
- one_adult: one adult member of the assessment unit must be eligible
- all_members (or all or taxunit): all members of the assessment unit must be eligible
- all_adults: all adult members of the assessment unit must be eligible
- nobody: calculations are carried out for each assessment unit, this is the default if the parameter is not indicated

"assessment unit" as well

as the concrete meaning of "adult" refers to the definition of the assessment unit indicated by the parameter TAX_UNIT

(which is described in more detail below). "eligible"

means that the variable sel_s

is set to one for this person (usually by using the function Elig). More precisely, it is not necessarily the variable sel_s,

which must be set to one, but the variable indicated by the parameter ***elig_var***. Example 2 illustrates the use of an alternative eligibility variable. If the parameter *elig_var* is not indicated the variable sel_s is used by

the parameter **who_must_be_elig** as a default.

The optional parameters **who_must_be_elig** and *elig_var* are provided by all policy functions (even Elig). None of the system functions provides them.

For special functions the descriptions of the respective functions in section EUROMOD Functions - Summary of functions and their parameters indicate whether this is the case or not.

Example 2:

Policy	**SL_demo**	*Comment*
Elig	**on**	**eligibility rule**
elig_cond	{dag > 80}	person must be aged older than 80
output_var	stm01_s	write "eligibility" to a temporary variable
TAX_UNIT	tu_individual_sl	
ArithOp	**on**	**benefit calculation**

who_must_be_elig	taxunit	if assessment unit is individual, parameter can be set to "taxunit/all" or "one" with the same result
elig_var	stm01_s	use same eligibility variable as above
formula	100#m	the benefit amounts to 100 € per month
output_var	sin01_s	write result to some intermediate variable
TAX_UNIT	tu_individual_sl	

Common

parameters limiting results

Three common parameters allow limiting the result of functions. These are **lowlim** for setting a lower limit, **uplim** for setting an upper limit and **threshold** to define a threshold. The limit parameters are always optional. They are provided by all policy functions except Elig and Allocate. None of the system functions provides them. The descriptions of the functions in section EUROMOD Functions - Summary of functions and their parameters indicate whether a special function provides the parameters or not. The examples below illustrate the usage of the limit parameters.

Example 3: Lower and upper limit

Policy	**SL_demo**	*Comment*
ArithOp	**on**	
formula	stm01_s	
lowlim	100#m	
uplim	1000#m	
output_var	stm02_s	
TAX_UNIT	tu_individual_sl	

In example 3 the output variable stm02_s (s=system, tm=temporary, _s=simulated) is set to 100 in all cases where the (input) variable stm01_s is smaller than 100 (lower limit applies). It is set to the value of stm01_s in all cases where stm01_s is between 100 and 1000 (no limit applies). And it is set to 1000 in all cases where stm01_s is greater than 1000 (upper limit applies).

Example 4: Threshold

Policy	**SL_demo**	*Comment*
ArithOp	**on**	
formula	stm01_s	
threshold	100#m	
output_var	stm02_s	
TAX_UNIT	tu_individual_sl	

In example 4 the output variable stm02_s is set to zero in all cases where the (input) variable stm01_s is smaller than 100 (threshold

applies) and to the value of stm01_s

in all other cases (threshold does not apply).

Example 5: Threshold and lower limit

Policy	**SL_demo**	*Comment*
ArithOp	**on**	
formula	stm01_s	
threshold	100#m	
lowlim	50#m	
output_var	stm02_s	
TAX_UNIT	tu_individual_sl	

Example 5 shows the combined use of

a threshold and a lower limit. In this case the output variable is not set to zero if it is below the threshold, but to the value of the lower limit. This means that stm02_s is set to 50 in all cases where

the (input) variable stm01_s is smaller than 100 and

to the value of stm01_s in all other cases.

For the sake of completeness another, very rarely used

common parameter, **limpriority**, should be mentioned. If there is a conflict between upper and lower limit, i.e. the upper limit is below the lower limit (which is nonsense if limits are defined by amounts, but may be the case if they are defined by variables or incomelists)

usually a warning is issued. This can be avoided by using the parameter limpriority. Possible values are *upper*, i.e.

the upper limit dominates in conflict cases, or *lower* i.e. the lower limit dominates. If the parameter is not defined and the warning ignored, the upper limit dominates.

The common parameter TAX_UNIT

The parameter **TAX_UNIT** allows for the definition of the assessment unit a function refers to. Assessment units range from individual units (each person builds her/his own unit) over various definitions of family units to household units (all persons of the household belong to the same unit). The possibility of defining the assessment unit not only on policy level, but on function level is one of the features, which make EUROMOD especially flexible. In fact it is even possible to change the assessment unit within a function. The other side of the coin is however, that it takes some learning effort and experience to understand the consequential complexity. The sections EUROMOD Functions - Parameter values and the assessment unit and EUROMOD Functions - The system functions DefTU and UpdateTU deal with these complexities. The former explains how parameters are interpreted if the assessment unit consists of more than one person and to which person within the unit the function result is assigned. The latter describes how an assessment unit is defined.

The parameter is compulsory for all policy functions, i.e.

must be indicated. System functions do not have a TAX_UNIT

parameter, except for the function DefOuput,

where output is printed on the level of the indicated assessment unit (see EUROMOD Functions - The system function DefOutput). Whether a special function has a TAX_UNIT parameter or not is indicated in the descriptions of the functions in section EUROMOD

Functions - Summary of functions and their parameters.

Common parameters controlling whether a function is processed

The **switch** is

another feature of functions, which is not literally a parameter. As the examples show, functions can be "switched off", i.e. they are skipped by the model run. In fact the switch has four possible states:

- **on** (or 1): function is switched on
- **off** (or 0): function is switched off
- **toggle** (or 2): function is initially switched off, but may be switched on later (using the function ChangeParam).

  toggle tells the model that

  the parameters of the function should be read and checked, which would not be the case if the function were switched off. For a more detailed explanation see section EUROMOD Functions - The special function ChangeParam.
- **n/a**: function is not applicable. With respect to treatment of the function by the model run this state is equal to the off state. However, it stresses that the function is not only switched off (for whatever reason) but has no meaning for the respective system. Moreover, the user interface uses the *n/a* state in its export/import functionalities to assess whether a function is relevant for a system (see Working with EUROMOD - Importing and exporting systems).

Finally, the parameter **run_cond** allows for a conditional processing of the function. That means the function is only carried out if the respective condition is fulfilled. The functionality of the parameter should not be confused with the task of the parameters affecting "eligibility" described above. The latter specify conditions, which are individual or household based and therewith determine whether a function is processed for a specific unit. In contrast, the parameter run_cond is a conditional switch. Consequently, differently from eligibility conditions, run conditions are not intended to be used with individual/household based operators.[1] Also note that, the output variable of a function, which does not fulfil the run condition, stays uninitialised (analogous to switched off functions). Typically the parameter run_cond is used in more advanced applications of the model, e.g. loops, where the condition refers to a specific processing state (e.g. iteration of the loop) or some other global condition (e.g. some total is reached/not reached). This optional parameter is available for all policy functions as well as several system functions (the function descriptions in section EUROMOD

Functions - Summary of functions and their parameters indicate for which).

[1] Note that the programme does issue a warning if individual/household specific operators are used with *run_cond*. If this warning is ignored, the condition is fulfilled, if the assessment unit fulfils the condition. Conditions which refer to a smaller unit are evaluated for the head of unit (respectively the unit she/he belongs to). As a consequence, taking this and the fact that the output variable stays uninitalised if the run condition is not fulfilled, *run_cond* could in principle be used as a shortcut instead of *elig*. This

is however considered as bad practise and only recommended for testing purposes.

Types of parameter values

Apart

from classifying parameters with respect to their functionality (i.e. affecting output, limiting result, etc.) or into compulsory/optional, there is another possible grouping. Parameters can be classified by the values they take as follows:

Yes/no parameters

Such parameters allow only for two values: yes (or 1) and no (or 0).

Amount parameters

Amount parameters take either monetary or non-monetary

amounts. Example 1 shows the tax schedule of a simple income tax policy. There are two bands: one for taxable income below 5,000 Euro annually and one for income above this amount. For the first band a tax of 10% of taxable income is due, for the second the rate is 25%. The parameters band_uplim and band_rate are amount parameters, where the former takes monetary amounts and the latter non-monetary amounts.[1]

Example 1:

Policy	*Grp/No*	**SL_demo**	*Comment*
SchedCalc		**on**	**tax schedule**
base		il_taxableY	
band_upLim	1	5000#y	for annual taxable income up to 5,000
band_rate	1	0.1	the tax amounts to 10% of taxable income
band_rate	2	0.25	and 25% for income above this amount
output_var		tin_s	
TAX_UNIT		tu_individual_sl	

Usually amount parameter values are

followed by their "period", for example the band limit of 5000 is followed by #y for yearly. EUROMOD internally converts all amounts to monthly, e.g. the 5000 are divided by 12. It is good practise to always indicate a period, though #m for monthly has

no real effect, as no conversion to monthly is necessary. It is however more transparent to explicitly state whether amounts are annual, monthly or some other period. Of course there are amounts where a period does not make sense, as for example for the two rate parameters or capital values.

The period can take on the following values:

- #m for monthly (no conversion)
- #y for yearly (divided by 12, more precisely multiplied by 0.08333333333333)
- #q for quarterly (divided by 3, more precisely multiplied by 0.3333333333333)
- #w for weekly (multiplied by 4.34 =365/12/7)
- #d for daily (multiplied by 30.5)
- #l for labour day (multiplied by 21.73)
- #s for labour day in a six days week (multiplied by 26.07)
- #c for capital (no conversion)

Moreover each period can be used with r for rate, e.g. #mr for monthly rate. Rules for conversion are the same.

Note that, despite of numbers, constants as defined

by func_DefConst can be used as amount parameter values.

Variable parameters

Variable parameters take EUROMOD

variables as values (see EUROMOD Basic Concepts - EUROMOD terminology).

In most of the examples we come across a very important variable parameter, the parameter output_var.

If variable parameters are used with an assessment unit that comprises more than one person the question arises how to interpret this. This issue is dealt with in section EUROMOD Functions -

Parameter values and the assessment unit.

Incomelist parameters

Incomelist parameters take EUROMOD

incomelists as values. Incomelists

are important EUROMOD concepts. They are usually defined in the special policy ILDef_cc of a country's parameter file (see EUROMOD Basic Concepts - Presentation of countries' tax-benefit-systems). Section EUROMOD Functions - The system function DefIL describes how to define an incomelist.

Generally speaking an incomelist is a variable, which

is composed of other variables, e.g. the incomelist il_earns may be the sum of the variables yem (y=income, em=employment) and yse

(se=self employment). The name of an incomelist by convention starts with il_ or ils_, where il_ is used for "normal" incomelists

and ils_ for system incomelists. The

latter are incomelists, which exist in all countries'

models and have a certain definition, for example ils_dispy is the incomelist for disposable income as defined in EUROMOD. In principle incomelists (once defined) are applied in the same way as variables, therefore there is roughly any parameter which takes only incomelists but not

variables. Which leads to the next type ...

Variable-incomelist parameters

A few parameters take as well variables as incomelists as values. As an example, the parameter *Share_Prop* of the function *Allocate* features this type.

Name parameters

Such parameters allow for indicating names, e.g. the name of

a file, incomelist, etc. As an example, the parameter

File of the function *DefOutput* is a *name* parameter, taking the name of the output file.

Query parameters

EUROMOD offers a couple of so-called "queries"

which allow for more complex questions, as for example how many children are there in the assessment unit. These queries are listed and described in the section EUROMOD Functions - Queries. The result of a query is either yes or no, e.g. for the query IsLoneParent, or some (monetary or non monetary) value, e.g. for the queries GetPartnerIncome respectively nDepChildrenInTu. Actually, there are no "pure" query parameters, i.e. parameters that only take queries as their values, rather queries are usually

used within formulas and conditions. Which leads to the next

two types ...

Formula parameters

Formula parameters are in fact little calculators. Example 2

shows a somewhat more complex application, by calculating a benefit for persons in education, amounting to 500 Euro monthly, supplemented by half of the expenditure on education. Any other education benefit is deducted from the resulting amount, where a lower limit ensures that no negative benefit results.

For determining whether a person is in education the formula applies a query (IsInEducation), the basic benefit is indicated as monetary amount (500#m), the supplement for expenditure on education is calculated by using a variable (xed)

and dividing it by a non monetary amount (2). Finally, an incomelist is used to determine other education benefits (il_OthEducBen).

Example 2:

Policy	**SL_demo**	*Comment*
ArithOp	**on**	

formula	IsInEducation * (500#m + xed / 2) – il_OthEducBen	
lowlim	0	
output_var	stm01_s	
TAX_UNIT	tu_individual_sl	

In general formula parameters take

amounts (monetary and non monetary), variables, incomelists and queries as operands and combine them by simple arithmetic operations to calculate some result. Consequently, formula parameters can be used as amount, variable, incomelist

and query parameters (by using just one operand of the respective type and no arithmetic operations). In fact, there are no pure query parameters and very few pure amount and incomlist parameters. Section EUROMOD Functions - The policy function ArithOp provides a more detailed description of the formula parameter.

Condition parameters

Similar to formula parameters condition parameters take

amounts, variables, incomelists and queries as

operands. However, they combine them by logical and comparison operators to evaluate a condition with a yes/no result. Example 3 shows a somewhat more complex application, which tests whether there is at least one dependent child in the family and if so, whether earnings are below 15,000 Euro annually or unemployment benefits are received. A more detailed description of condition parameters is provided in the section EUROMOD Functions - The policy function Elig.

Example 3:

Policy	**SL_demo**	*Comment*
Elig	**on**	
elig_cond	{nDepChildrenInTu>1} & ({il_earns<15000#y} | {bun>0})	
TAX_UNIT	tu_sben_family_sl	

Taxunit parameters

A few parameters take assessment units as their values, the most important is the parameter TAX_UNIT, which is described in more detail in section EUROMOD Functions - Common Parameters.

The name of assessment units by convention starts with tu_.

Categorical parameters

Some parameters take a limited number of defined values, as

for example the parameter who_must_be_elig,

taking the values one_member

/ one_adult / all_members / all_adults (also see section EUROMOD Functions - Common Parameters).

Footnote parameters

Finally, there are parameters, which serve the further

specification of other parameters. They can be easily identified by names starting with the character #, e.g. #_amount. The application of such parameters is described in detail in section EUROMOD Functions -

Footnote parameters for the further specification of operands.

[1]

Actually, the parameters are formula parameters. They are

used for exemplification here because there are rarely any pure amount parameters.

Parameter values and the assessment unit

Assessing the result of a function

If parameters taking variables, incomelists

and queries as their values are used with assessment units, which comprise more than one person, there is an issue of interpretation. Example 1 illustrates the problem. The assessment unit of the function is the whole household. So what does IsDisabled mean? Is

the condition fulfilled if there is one disabled in the household, or must all household members be disabled, or a special person? The next question is, to whose earnings does the incomelist il_earns refer? To all earnings within the household? To

the earnings of the disabled person? Similar questions could be posed with respect to the housing costs variable xhc.

Example 1:

Policy	**SL_demo**	*Comment*
ArithOp	**on**	
formula	IsDisabled * xhc – il_earns	benefit funds expenditure for housing
lowlim	0	if the head of the household is disabled
output_var	bho_s	any earnings (received by anyone in the household)
TAX_UNIT	tu_household_sl	are deducted

To answer these questions lets first state the general rules.

Level of Interpretation	**... used in condition parameters**	**... used in other parameters**
monetary variables and incomelists ...	assessment unit	assessment unit
non-monetary variables and individual level queries ...	individual	head of assessment unit
non individual level queries ...	consult description in section EUROMOD Functions - Queries	consult description in section EUROMOD Functions - Queries

For the clear interpretation of

these rules some further information is necessary:

- "assessment unit"

  refers to the assessment unit defined by the parameter TAX_UNIT

  of the function and means that values are added up over all members of the unit.

- Who is the head of the assessment unit is

  determined on the basis of the respective definition via the function DefTU. (See section EUROMOD Functions - The system functions DefTU and UpdateTU for information with respect to defining an assessment unit.)

- Special care needs to be taken if monetary

  variables and incomelists are used in any of the

  condition parameters of the function DefTU

  (e.g. DepChildCond). The

  rule that they are interpreted on assessment unit level is still valid.

  However, as the model is operating on a not yet finally defined assessment unit, it is not clear what this means, if the assessment unit is of type *SUBGROUP*.[1] Therefore, it is good practise in such cases to always explicitly define the level of assessment by using a level parameter (see section EUROMOD Functions - Footnote parameters for the further specification of operands*)*.

- Whether a variable is monetary or non-monetary

  is defined in the variable description file. Note that simulated temporary (stmxx_s) and intermediate (sinxx_s) variables are defined to be monetary.

- It is possible to change the assessment unit

  generally used by the function (i.e. indicated by parameter TAX_UNIT) for single variables, incomelists or queries. As examples for the application of this feature consider a personal level benefit, where a means test refers to family income or a family based benefit, where a means test should only cover parents’ income. A detailed description of this feature is provided in section EUROMOD Functions - Footnote parameters for the further specification of operands.

Before answering the question

raised above, lets exercise by interpreting some more

simple and stylised examples, based on a simple sample household.

idperson	yem	dag	dms	ils_origy	is head?
101	2000	40	2	2000	yes
102	1500	38	2	1800	no
103	0	7	0	0	no

Lets

interpret some formula parameters of the function ArithOp (i.e. "other" parameters with respect to the table of rules above) assuming that the assessment unit (parameter TAX_UNIT) is the whole household.

formula	result	interpretation
yem*2	(2000+1500+0)*2=7000	**yem** is a monetary variable therefore added up over all members of the assessment unit
ils_origy/2	(2000+1800+0)/2=1900	**ils_origy** is an incomelist therefore added up over all members of the assessment unit
dag	40	**dag** (i.e. age) is a non-monetary variable therefore taken from the head of the assessment unit
IsMarried	1 (i.e. yes)	**IsMarried** is an individual level query therefore interpreted for the head of the assessment unit (dms=2 means married)

Now lets

interpret some elig_cond

parameters of the function Elig

(i.e. condition parameters with respect to the table of rules above) again assuming that the assessment unit is the whole household. As condition parameters produce a result on individual level, we need a second function and parameter to obtain a result on assessment unit level – in the example the parameter who_must_be_elig (wmbe).

elig_cond	individual result		assessment unit result		interpretation
			wmbe=one	wmbe=all	
{yem<1700}	101	(2000+1500+0)<1700=0	(0 or 0 or 0)=0	(0 and 0 and 0)=0	**yem** is a monetary variable therefore added up over all members of the assessment unit
	102	(2000+1500+0)<1700=0			
	103	(2000+1500+0)<1700=0			
{ils_origy>3000}	101	(2000+1800+0)>3000=1	(1 or 1 or 1)=1	(1 and 1 and 1)=1	**ils_origy** is an incomelist therefore added up over all members of the assessment unit
	102	(2000+1800+0)>3000=1			
	103	(2000+1800+0)>3000=1			

{dag<30}	101	40<30=0	(0 or 0 or 1)=1	(0 and 0 and 1)=0	**dag** (i.e. age) is a non-monetary variable therefore interpreted on individual level
	102	38<30=0			
	103	7<30=0			
{IsMarried}	101	(dms=2)=1	(1 or 1 or 0)=1	(1 and 1 and 0)=0	**IsMarried** is an individual level query therefore interpreted on individual level (dms=2 means married)
	102	(dms=2)=1			
	103	(dms=2)=0			

After these exercises we are fit to

answer the questions raised above. IsDisabled

is an individual level query, i.e. it asks whether the head of the household (which is the assessment unit of the function) is disabled. The housing costs variable xhc is monetary,

i.e. it is interpreted on household level. The incomelist

il_earns is interpreted on

household level as well. Summarising, the benefit funds the household's expenditure for housing, if the head of the household is disabled. Any earnings, received by anyone in the household, are deducted.

Storing the result of a function

So far we have clarified how the result of a function is

evaluated, but it is still not clear where this result is stored. Taking example 1 it is clear that it should be stored in the variable bho_s (b=benefit, ho=housing, _s=simulated). But in the variable bho_s of which of

the household members? The following rules answer this question and clarify what happens with the function result in general.

The result of a function is assigned to the head of the assessment unit. That means:

- If parameter **output_var** is used the variable defined by parameter **output_var** is set to (overwritten by) the function result.
- If parameter **output_add_var** is used the function result is added to the variable defined by parameter **output_add_var**.

- Any variable defined by parameter **result_var** is set to (overwritten by) the function result.

For all other members of the assessment unit the following applies:

- If parameter **output_var** is used the variable defined by parameter **output_var** is set to zero.
- If parameter **output_add_var** is used the variable defined by parameter **output_var** is not changed (i.e. keeps its value).[2]
- Any variable defined by parameter **result_var** is set to zero.

For all members of

not eligible assessment units (with respect to the settings of parameter who_must_be_elig) the same rules concerning output and result variable apply as for non-head members of eligible assessment units.

The following stylised example may illustrate the rules,

assuming the result to store is 1000 and the assessment unit of the function is the household.

idperson	is head?	is eligible unit	previous value of output variable	value of output variable if		value of result variable
				parameter *output_var* is used	parameter *output_add_var* is used	
101	yes	yes	undefined	1000	1000	1000
102	no		undefined	0	0	0
201	yes	yes	1500	1000	2500	1000
202	no		1500	0	1500	0
301	yes	no	undefined	0	0	0
302	no		undefined	0	0	0
401	yes	no	1500	0	1500	0
402	no		1500	0	1500	0

The rules apply for all policy functions with two exceptions. The function Allocate (see section EUROMOD

Functions - The policy function Allocate) is designed to reallocate the value of variables, i.e. allows for assigning results to someone else than the head.

As a consequence it must be an exception of the rule. Also the function Elig shows a different behaviour, which is explained in detail in section EUROMOD Functions - The policy function Elig.

[1] For *DepChildCond* the default assessment unit is the whole household, but for any other condition a rather arbitrary subgroup is used.

[2] Except the variable was "undefined" before – in which case it is set to zero (a defined value). Note that EUROMOD initialises all simulated variables by a value called VOID, which amounts to 0.0000000000001, to mark them as undefined.

Footnote parameters for the further specification of operands

The operands of formula parameters and condition parameters may be "further specified" by using so called footnotes and respective footnote parameters. Explanations and examples below illustrate what this means and introduce another bunch of common parameters, as footnote parameters are applicable with several functions (more specific, with all functions providing formula and/or condition parameters[1]).

Further specification of operands by footnotes: limits

Besides showing how to apply limits not only on the function result (as by parameters lowlim,

uplim and threshold) but also on single operands, example 1

illustrates the usage of footnotes.

Example 1:

Policy	*Grp/No*	**SL_demo**	*Comment*
func_ArithOp		**on**	
formula		yem#1 * 0.1 + yse#2 * 0.15	10% of employment and 15% of self-employment income
#_uplim	1	20000#y	upper limit for employment income
#_uplim	2	30000#y	upper limit for self-employment income
output_var		sin01_s	
TAX_UNIT		tu_individual_sl	

The example calculates 10% of

employment income (yem)

and 15% of self-employment income (yse)

as the function's result, where an upper limit of 20,000 Euro annually is set for employment income and an upper limit of 30,000 Euro annually for self-employment income. The #1 and #2 in the formula indicate, that the operands yem and yse are to be further specified, while the parameters #_uplim, *Grp/No 1* and #_uplim, *Grp/No 2* conduct this further specification, in this case by defining the mentioned limits.

Note that, footnotes can apply any integer number, e.g.

using #4711 and #_uplim, *Grp/No 4711* instead of #1 and #_uplim, *Grp/No 1* would be possible as well. Also note that, footnotes can be used more than once: if the upper limit for employment income and self-employment income were the same, one could write the formula as yem#1*0.1+yse#1*0.15

and omit the parameter #_uplim,

Grp/No 2.

Further specification of operands by footnotes: amounts

If there is more than one system it is sometimes more transparent to indicate amounts outside the formula.

Example 2 illustrates the issue.

Example 2:

Policy	*Grp/No*	**SL_demo**	**SL_reform**	*Comment*
func_ArithOp		**on**	**on**	
formula		amount#1 + yem*amount#2	amount#1 + yem*amount#2	function results to
#_amount	1	10000#y	15000#y	a basic amount
#_amount	2	0.1	0.15	plus x% of empl. income
output_var		sin01_s	sin01_s	
TAX_UNIT		tu_individual_sl	tu_individual_sl	

It would as well be possible to

write the formulas as 10000#y + yem*0.1 for the system SL_demo and 15000#y + yem*0.15 for SL_reform, but with this approach the differences between the base and the reform system would be less transparent. Apparently, this matter becomes especially significant, with a rather complex formula and the implementation of several policy years. Therefore, good modelling practise suggests to thoroughly considering, whether to pack amounts, which tend to change with the policy year within the formula or outside.

Further specification of operands by footnotes: assessment units

Section EUROMOD

Functions - Parameter values and the assessment unit discussed the interpretation of operands (variables, incomelists and queries) if they are used with assessment units, which comprise more than one person, and announced the possibility of changing the function's assessment unit (indicated by parameter TAX_UNIT) for single

operands. Example 3 now illustrates this possibility.

Example 3:

Policy	*Grp/No*	**SL_demo**	*Comment*
func_ArithOp		**on**	
formula		IsInEducation * xed – ils_dispy#1 / 3	individual level education benefit
#_level	1	tu_sben_family_sl	financing expenditure on education
lowlim			with means test on family level
output_var		bun_s	
TAX_UNIT		tu_individual_sl	
func_ArithOp		**off**	
formula		IsInEducation#1 * xed#1 – ils_dispy / 3	
#_level	1	tu_individual_sl	does not work !!!
lowlim		0	
output_var		bun_s	
TAX_UNIT		tu_sben_family_sl	

Both functions in example 3 seem to

calculate a benefit, which finances people's expenditure on education (xed). The assessment unit of the first function is individual. That means, if the footnote was ignored, all education expenditure of a person in education going beyond a third of her/his personal disposable income (ils_dispy)

would be financed. With the footnote #1 applied on the operand ils_dispy and

the specifying parameter #_level, Grp/No

1 defining tu_sben_family_sl

as the relevant assessment unit, the means test is extended to family's disposable income, i.e. only education expenditure going beyond a third of the family's disposable income is financed. Note, that the level of education expenditure is not changed, i.e. the benefit still concerns the person's education expenditure.

The second function looks like doing the same thing the other way round, but that's not quite true. In fact it would lead to the following warning – therefore it is switched off: "Assessment unit cannot be used as alternative level. (Only assessment units containing the function's main assessment unit are allowed ...); Handling:

Alternative level is ignored". The main assessment unit of the function is *tu_sben_family_sl*,

i.e. some definition of the family. Thus the warning says that only assessment units containing the family are allowed, and obviously an individual’s family is not part of the individual (but the other way round). But why is this problematic?

The answer is, that the programme would not know *which* individual in the family and therefore prefers to ignore the level change over possibly causing confusion by doing something arbitrary. That means level changes are only possible to bigger (comprising) units (in the example e.g. the whole household) but not to smaller (sub) units.

This bad example demonstrates the clubfoot of the high flexibility offered by adaptable definitions of assessment units in general and by the possibility of applying them on the most detailed level in particular, and suggests using these options with great care, especially with respect to transparency.

Further specification of operands by footnotes: specification of queries

A few queries need or allow for further specification. For example for the query GetPartnerIncome a specification of "income" is necessary. Example 4 illustrates how such specifications are established.

Example 4:

Policy	*Grp/No*	**SL_demo**	*Comment*
func_ArithOp		**on**	
formula		nDepChildrenInTaxunit#1	count the number of dependent children
#_AgeMin	1	10	aged 10 to 16
#_AgeMax	2	16	
output_var		stm01_s	
TAX_UNIT		tu_sben_family_sl	

The example calculates the number of

dependent children aged between 10 and 16 in the family. nDepChildrenInTaxunit is a query that counts the dependent children in the assessment unit. It has two optional parameters, #_AgeMin and

#_AgeMax, which are set to

10 respectively 16 in the example. To find out which queries require or allow for further specification see section EUROMOD

Functions - Queries.

Further specification of operands by footnotes: types

For the sake of completeness the following example illustrates a rarely applicable usage of footnotes. In principle EUROMOD automatically finds out whether a non-numeric operand is a variable, an incomelist or a query. If there is however an ambiguity because a variable and an incomelist have the same name, the type of an operand can be explicitly defined. As such equal naming is in fact bad modelling style (not least as names of incomelists should always start

with il

or ils) the usage of this

functionality should be exceptional. Example 5 shows how to (unnecessarily) define yem as a variable.

Allowed parameter values are var

(or variable), il (or incomelist)

and query.

Example 5:

Policy	*Grp/No*	**SL_demo**	*Comment*
func_ArithOp		**on**	
formula		yem#1 * 0.1	
#_type	1	var	
output_var		stm01_s	
TAX_UNIT		tu_individual_sl	

[1] Note that in conditions footnotes can be placed either inside the curly brackets, i.e. *{A#x}* or outside *{A}#x*. Internally the model changes *{A}#x* to *{A#x}*, respectively *{A < B}#x* to *{A#x < b#x}*.

Identifiers and the placeholders =cc= and =sys=

Placeholders

EUROMOD add-ons are commonly designed for being used with not just one country, but with several or all countries. (For more information on add-ons see EUROMOD Functions - EUROMOD add-ons and the special functions AddOn_Applic, AddOn_Pol, AddOn_Func and AddOnPar.) In implementing them it is sometimes necessary to refer to components of the tax-benefit implementations, which are in principle common, but have country- or system-dependent specifications. As a typical example, all countries have some definition of the assessment unit household, but specifications (e.g. who is a child) differ, therefore the respective assessment units are called tu_household_ee, tu_household_uk, etc. To be still able to refer to them all at once the placeholder =cc= can be used, e.g. tu_household_=cc=. The model will replace the placeholders by the respective country acronym at runtime. As another typical example an add-on may produce a special output. In order to not overwrite this output if the add-on is run for several systems, one may want to us *someoutput_=sys=* as filename (parameter *File* of function *DefOutput*). Again the model will replace the placeholder by the respective system name at runtime, e.g. *someoutput_UK_2012*. In principle the placeholders not only can be used with add-ons but in each parameter file (though probably seldom necessary).

Identifiers

Each EUROMOD policy, function and parameter possesses a unique identifier in the form of a GUID[1]. More precisely, identifiers are system specific, which means that each policy, function or parameter possesses one unique identifier per system. These unique identifiers are assigned once a component is created and kept for the whole lifetime of the component.

In addition each policy, function and parameter possesses a "symbolic identifier". The symbolic identifier of policies is simply their name. The symbolic identifier of a function is composed of the name of the policy where the function is located and the order of the function, separated by _#. For example *tin_es_#3* refers to the third function in the Spanish income tax policy (*tin_es*). The symbolic identifier of a parameter is composed of the name of the policy where the parameter is located, the order of the function containing the parameter and the order of the parameter itself, the former order again separated by _# and the latter by a point. For example *tin_es_#3.7* refers to the seventh parameter in the third function of the Spanish income tax policy.

Symbolic identifiers are constructed somewhat different for the functions *DefIL* and *DefTU*. Instead of the order of the function the incomelist's or taxunit's name is used. For example *ildef_bg_#ils_dispy* refers to the function defining disposable income for Bulgaria (more precisely the *DefIL* function in the policy *ildef_bg* with parameter *name* set to *ils_dispy*). Similarily *tudef_bg_#tu_household_bg.3* refers to the third parameter in the function defining the Bulgarian household concept (*tu_household_bg*). This sort of referencing allows an add-on for example to add components to incomelists which exist for all countries, e.g. a benefit or tax calculated by the add on can be made part of standard disposable income.

Due to their construction symbolic identifiers are not system specific and, though unique at a certain time, may change with alterations of the order of functions or parameters or the renaming of a policy. The reason for still preferring them over the more secure unique identifiers is similar to the usage of placeholders, i.e. they are not supposed to refer to something unique but to several countries or systems.

Identifiers are used by several functions, most frequently by the *AddOn_xxx* functions, but also by the loop functions and *ChangeParam*. For an example see EUROMOD Functions - EUROMOD add-ons and the special functions AddOn_Applic, AddOn_Pol, AddOn_Func and AddOnPar. **Please note that symbolic identifiers can only be used within add-ons!** Within country implementations please use unique identifiers.

To use an identifier right click the respective policy, function or parameter and select either *Copy Identifier* or *Copy Symbolic Identifier* from the context menu. This copies the respective identifier to the clipboard, from where it can be pasted to the required location.

If you prefer the "manual" way to create identifiers, this summary of syntax may help:
General: PolicyName[_#FunctionNumber][.ParameterNumber] (examples: tin_es, tin_es_#3, tin_es_#3.7)
DefIL: PolicyName_#IncomelistName[.ParameterNumber] (examples: ildef_bg_#ils_dispy, ildef_bg_#ils_dispy.6)

DefTU: PolicyName_#TaxunitName[.ParameterNumber] (examples: tudef_bg_#tu_household_bg, tudef_bg_#tu_household_bg.3)

[1] GUID stands for Globally Unique IDentifier, which is a unique reference number used as an identifier in computer software. The value of a GUID is represented as a 32-character hexadecimal string.

Description of functions and their parameters

This section provides a descriptive explanation of EUROMOD functions with many examples. For a full description of the functions' parameters see EUROMOD Functions - Summary of functions and their parameters.

The policy function ArithOp

Basically, ArithOp offers the functionality of a simple calculator.

It provides, apart from the common parameters, just one parameter, formula, containing the formula to be calculated to derive

the function's result. For examples see sections EUROMOD

Functions - What are functions and how are they used and EUROMOD Functions - Common parameters and

general features in interpreting their values.

The formula allows for the following

operations:

- addition: operator +
- subtraction: operator -
- multiplication: operator *
- division: operator /
- remainder of division: operator \, e.g. 22\5 (result: 2)
- raising to a power: operator ^, e.g. 2 ^ 3 (result: 8)
- percentage: operator %, e.g. yem*3% (result: yem*(3/100))
- minimum and maximum: operators <min> and <max>, e.g. 10 <min> 15 (result: 10) *(deprecated in EM3)*
- absolute value: operator <abs>(), e.g. <abs>(-22) (result: 22), <abs>(50-70) (result: 20) *(deprecated in EM3)*
- negation: operator !, e.g. !IsMarried, !(17) (result: 0), !(0) (result: 1)
- logical operators: operators & and |, e.g. IsMarried & IsUnemployed | IsDisabled
- comparison operators: operators <, >, =, <=, >=, !=, e.g. dag >= 18
- inline functions:
  - ABS(1): Absolute value, e.g. abs(-22) (result: 22), abs(50-70) (result: 20) *(only available in EM3)*

    - MIN(2): Minimum value, e.g. min(10, 15) (result: 10) *(only available in EM3)*
    - MAX(2): Maximum value, e.g. max(10, 15) (result: 15) *(only available in EM3)*
    - LN(1): Natural logarithm, e.g. ln(10) (result: 2.30258509299) *(only available in EM3)*
    - LOG(1-2): Logarithm, e.g. log(10, 3) (result: 1.01283722471), log(10) (result: 1) *(only available in EM3)*
    - LOG10(1): Logarithm base 10, e.g. LOG10(10) (result: 1) *(only available in EM3)*
    - EXP(1): Exponential, e.g. exp(3.14) (result: 23.1038668587 *(only available in EM3)*
    - SQRT(1): Square root, e.g. sqrt(4) (result: 2) *(only available in EM3)*
    - CEILING(1): Ceiling value, e.g. ceiling(3.1) (result: 4), ceiling(3.9) (result: 4) *(only available in EM3)*
    - FLOOR(1):Floor value, e.g. floor(3.1) (result: 3), floor(3.9) (result: 3) *(only available in EM3)*)
    - ROUND(1-2):Round value, e.g. round(3.1) (result: 3), round(3.9) (result: 4), round(3.375, 2) (result: 3.38) *(only available in EM3)*
    - POWER(2): Raising to a power, e.g. power(2, 3) (result: 8) *(only available in EM3)*

to be used with the following operands:

- numeric values, e.g. 10, 0.3, -25
- numeric values with a period, e.g. 12000#m, 1000#y (see EUROMOD Functions - Types of parameter values)
- amount#i as place holders for numeric values specified by footnote parameters (see EUROMOD Functions - Footnote parameters for the further specification of operands)
- variables, e.g. yem (see EUROMOD Basic Concepts - EUROMOD terminology)

- incomelists, e.g. ils_dispy (see EUROMOD Basic Concepts - EUROMOD terminology)
- queries, e.g. IsUnemployed (see EUROMOD Functions - Queries)
- random numbers, rand (see EUROMOD Functions - The special function RandSeed)

Order of operation rules:

- variables, literals, percentage %
- before parenthesis (), inline functions
- before <min>, <max>, <abs>, !
- before power ^
- before multiplicative operations *, /, \
- before additive operations +, -
- before comparison operations <, >, =, <=, >=, !=
- before logical and &
- before logical or |

The policy function Elig

The

function Elig is often referred to as eligibility

function, because it is most frequently used for determining the eligibility for receiving a certain benefit. More general, its purpose is to implement conditions. Usually such conditions evaluate whether a certain assessment unit is eligible for receiving a benefit / liable for paying a tax. The basic use of Elig is discussed in sections EUROMOD Functions - Interactions between functions and EUROMOD

Functions - Common Parameters. Briefly recapitulated, the function sets a variable (usually sel_s) to zero or one, based on a

condition defined by the parameter elig_cond.

Subsequent functions use this information, together with the setting of the parameter who_must_be_elig, to determine whether

their calculations should be carried out for a certain assessment unit or not.

Syntax of the eligibility condition

Similar to the parameter formula of *ArithOp*, the parameter elig_cond of Elig

takes a "formula" to calculate the function's result, however, the result is restricted to the values false (zero) and true (non zero). As the parameter formula, the parameter elig_cond requires a certain syntax, it consists of:

- conditions which must be fulfilled, e.g. {dag<19}
- conditions which must not be fulfilled, e.g. !{ils_dispy>500#m}[1]
- combined by the logical operators & (and) and | (or), e.g. {dag<19} & !{ils_dispy>500#m}

Parentheses can be used to group conditions.

A single condition {...} has

- either one component, usually a yes/no query, e.g. {IsUnemployed} (see

EUROMOD Functions - Queries)

- or two components separated by a comparison operator >, <, >=, <=, = or !=, e.g. {poa=0}

The operands left and right from the comparison operator are (the same as for parameter formula of *ArithOp*):

- numeric values, e.g. 10, 0.3, (-25), note that negative values must be bracketed
- numeric values with a period, e.g. 12000#m, 1000#y (see EUROMOD Functions - Types of parameter values)
- amount#i as place holders for numeric values specified by footnote parameters (see EUROMOD Functions - Footnote parameters for the further specification of operands)
- variables, e.g. yem (see EUROMOD Basic Concepts - EUROMOD terminology)
- incomelists, e.g. ils_dispy (see EUROMOD Basic Concepts - EUROMOD terminology)
- queries, e.g. IsUnemployed (see EUROMOD Functions - Queries)

Interpreting conditions with respect to the assessment unit

As briefly

denoted in section EUROMOD Functions - Parameter values and the assessment unit, the function Elig

shows a somewhat different behaviour in determining parameter values with respect to the assessment unit and in setting the output variable.

- **Firstly, the output variable** (usually sel_s) **is individually set for each person in the assessment unit** (instead of the head of the unit as for other functions)**.**
- **The output variable is set to 1 if**

  (a) a person fulfils all personal conditions and (b) the assessment unit the person belongs to fulfils the assessment unit conditions, as defined

by parameter elig_cond.

Example 1 may help to understand these rules.

Example 1:

Policy	**SL_demo**	*Comment*
Elig	**on**	
elig_cond	{nDepChildrenInTaxunit>2} & {dag<10} & {ils_dispy<20000#y}	
TAX_UNIT	sben_family_sl	
ArithOp	**on**	
who_must_be_elig	one_member	
formula	100#m	
output_var	bfa_s	
TAX_UNIT	sben_family_sl	

The benefit of monthly 100 Euro is received by families with more than two dependent children, where at least one is younger than ten and family's disposable income is below 20,000 Euro annually. Let's figure out why. elig_cond consists of three conditions: {nDepChildrenInTaxunit>2}

clearly can be assessed on assessment unit level

only. {ils_dispy<20000#y} is also an

assessment unit level condition, because we know from section EUROMOD

Functions - Parameter values and the assessment unit that incomelists, if not defined elsewise, are assessed for the assessment unit. {dag<10}, on the contrary, is clearly an individual level condition. With respect to the rules outlined above, the output variable of Elig, i.e. sel_s, is set to one for all children aged less than 10 years, living in familys with more than two dependent children and disposable income below 20,000 Euro annually. Those children fulfil the individual level condition of being younger than ten and their family fulfils the two assessment unit level conditions. As the parameter who_must_be_elig states that at least one member of the assessment unit must be eligible, all families containing such a child receive the benefit.

The example (hopefully) illustrated why the function Elig needs to be an exception with respect to setting its output variable. In general, it could be stated that flexibility with respect to the assessment unit is especially useful for evaluating (eligibility) conditions. Therefore the possibility of changing the assessment unit for single operands is much more likely to be used with condition parameters than with formula parameters. Example 2 exemplifies such a use of an alternative assessment unit, by changing the means test of example 1.

Example 2:

Policy	*Grp/No*	**SL_demo**	*Comment*
Elig		**on**	
elig_cond		{nDepChildrenInTaxunit>2} & {dag<10} & {ils_dispy#1<20000#y}	
#_level	1	household_sl	
TAX_UNIT		sben_family_sl	
ArithOp		**on**	
who_must_be_elig		one_member	
formula		100#m	
output_var		bfa_s	
TAX_UNIT		sben_family_sl	

The means test now refers to whole household's disposable income instead of the disposable income of the family.

[1] Please note, that the not-operator !

only works on single conditions, e.g. !({IsDisabled} & {IsUnemployed}), is incorrect syntax.

The policy function BenCalc

The

function BenCalc is often referred to as the benefit calculator, as it allows for modelling a wide range of policy instruments, in particular benefits. This is accomplished by combining the functionalities of the functions *Elig* and *ArithOp*.

Basically, the function calculates its result as a sum of "components", where the value of a component is only added if a certain condition is fulfilled by at least one member of the assessment unit. The following stylised formulas illustrates the approach:

result = Sum$_i$ (comp$_i$_perTU if comp$_i$_cond = true) result = Sum$_i$ (comp$_i$_perElig * nElig if comp$_i$_cond = true) That means,

a component is only added if the component's condition is fulfilled, which is defined by the parameter compi_cond. This parameter follows the same rules as the parameter elig_cond of the function *Elig*.

The value of the component is either defined by the parameter compi_perTU or compi_perElig, which follow the same rules as the parameter formula of the function *ArithOp*.

In the former case simply the value as defined by compi_perTU

is added, whereas in the latter case the value as defined by compi_perElig multiplied by the number of assessment unit members fulfilling the condition is added. Example 1 illustrates the approach by modelling a simple child benefit, where each family with dependent children {nDepChildrenInTaxunit>0} receives a monthly amount of 100 Euro (comp_perTU, GrpNo 1)

plus monthly 10 Euro (comp2_perElig, GrpNo 2) for

any child younger than three {dag<3}.

Example 1:

Policy	*Grp/No*	**SL_demo**	*Comment*
BenCalc		**on**	
comp_cond	1	{nDepChildrenInTaxunit>0}	
comp_perTU	1	100#m	benefit pays a fixed amount if there are children
comp_cond	2	{dag<3}	

comp_perElig	2	10#m	plus an amount per child younger than 3
output_var		bch_s	
TAX_UNIT		sben_family_sl	

Example 2 presents a frequent application of the function BenCalc – a child benefit where the amount depends on the number of children.

Example 2:

Policy	*Grp/No*	**SL_demo**	*Comment*
BenCalc		**on**	
base		50#m	
comp_cond	1	{isNtoMchild}#1	
#_N	1	1	
#_M	1	2	
comp_perElig	1	$base	50 for the 1st and 2nd child
comp_cond	2	{isNtoMchild}#2	
#_N	2	3	
#_M	2	4	
comp_perElig	2	$base*1.5	75 for the 3rd and 4th child
comp_cond	3	{isNtoMchild}#3	
#_N	3	5	
#_M	3	99	
comp_perElig	3	$base*2	100 for each child beyond the 4th
output_var		bch_s	
TAX_UNIT		sben_family_sl	

The function result is calculated as the sum of three components. The condition for component one, comp_cond, Grp/No 1, is fulfilled if a person "is the Nth to Mth child", where N is defined by the parameter #_N, Grp/No 1 and M is defined by the parameter #_M, Grp/No 1, i.e. the person must be the 1st to 2nd child to fulfil the condition. comp_perElig, Grp/No 1 is set to $base a basic benefit

amount of monthly 50 Euro, defined by the parameter base.

That means 50 Euro monthly are paid for each potential first and second child.

In the same way component two defines that 75 Euro monthly (50*1.5) are paid for the third and fourth chid and component three adds 100 Euro monthly (50*2) for each fifth and any further child.

Example 3 uses BenCalc to model a child

benefit where the amount depends on the age of the children.

Example 3:

Policy	*Grp/No*	**SL_demo**	*Comment*
BenCalc		**on**	
base		50#m	
comp_cond	1	{IsDepChild} & {dag<=3}	
comp_perElig	1	$base*2	100 for children younger than 3
comp_cond	2	{IsDepChild} & {dag=4} & {dag<=14}	
comp_perElig	2	$base	50 for children from 4 to 14
comp_cond	3	{IsDepChild} & {dag>=15} & {dag<=20}	
comp_perElig	3	$base*1.5	75 for children from 15 to 20
output_var		bch_s	
TAX_UNIT		sben_family_sl	

The function result is calculated as the sum of three components. The condition for component one, comp_cond, GrpNo 1, is fulfilled by children who are up to three years old. The amount paid per child fulfilling the condition is defined by parameter comp_perElig, GrpNo 1 and set to $base*2, i.e. 100 Euro

monthly (50*2). In the same way an amount of 50 Euro monthly is paid for each child aged four to 14 years and an amount of 75 Euro monthly (50*1.5) for each child aged 15 to 20 years.

Example 4 shows a further typical application of the function BenCalc.

Example 4:

Policy	*Grp/No*	**SL_demo**	*Comment*
BenCalc		**on**	
base		100#m	
comp_cond	1	{IsHeadofTU}	
comp_perElig	1	$base	100 for the head
comp_cond	2	{IsPartnerOfHeadofTU}	
comp_perElig	2	$base*0.5	50 for the partner of the head
comp_cond	3	{IsDepChild}	
comp_perElig	3	$base*0.3	30 per dependent child
comp_cond	4	{IsDepParent}	
comp_perElig	4	$base*0.3	30 per dependent (grand)parent
output_var		bfa_s	
TAX_UNIT		big_family_sl	

Each assessment unit receives a monthly base amount of 100 Euro via component 1 – the condition is being the head of the assessment unit and obviously each assessment unit has exactly one. Component 2 adds 50 Euro

($base*0.5) for the partner of the head. Another 30 Euro per dependent child are added by component 3. And finally, component 4 adds further 30 for each dependent (grand)parent.

BenCalc provides parameters allowing for

the withdrawal of a benefit when income rises. Example 5 illustrates how they work.

Example 5:

Policy	*Grp/No*	**SL_demo**	*Comment*
BenCalc		**on**	
comp_cond	1	{IsLoneParent}	
comp_perElig	1	100#m	100 for lone parents with no employment income
withdraw_base		yem	
withdraw_rate		0.1	10 cents are withdrawn for each Euro earned
output_var		bchlp_s	
TAX_UNIT		individual_sl	

In this simple example lone parents with no employment income receive a monthly benefit of 100 Euro. Any employment income leads to a withdrawal of the benefit. However, the benefit is not withdrawn at a one to one rate, i.e. for each earned Euro one Euro of benefit is lost, but at a one to one tenth rate, i.e. for each earned Euro only 10 Cents of benefit are lost. The benefit is zero only at a monthly employment income of 1,000 Euro or above. The formula is: benefit (sum of components) minus withdraw_base times withdraw_rate. It is not necessary to define a lower limit of zero for the benefit (using parameter lowlim) to avoid negative results for employment income above 1,000, as the lower limit of the function result is automatically set to zero if the withdraw parameters are used.

There are two other parameters allowing for an alternative approach to withdraw a benefit with rising income. They are presented in example 6.

Example 6:

Policy	*Grp/No*	**SL_demo**	*Comment*
BenCalc		**on**	
comp_cond	1	{IsLoneParent}	

comp_perElig	1	100#m	100 for lone parents with no employment income
withdraw_base		yem	
withdraw_start		500#m	10 cents are withdrawn for each Euro earned
withdraw_end		1000#m	but only if income is below 100
output_var		bchlp_s	
TAX_UNIT		individual_sl	

Still lone parents receive a monthly benefit of 100 Euro, which is totally withdrawn at an employment income of 1,000 Euro monthly (withdraw_end). However, the withdrawal only starts at an income of monthly 500 Euro (withdraw_start), up to these earnings the full 100 Euro are received. To provide a "smooth"

decrease of the benefit, the withdrawal must be faster as in example 5, as it starts later. The implicit withdrawal rate is therefore one to one fifth, i.e.

20 Cent for each earned Euro beyond 500. The possibility of calculating an implicit withdrawal rate suggests that parameters withdraw_rate and withdraw_end are exchangeable – that's true.

Precisely the formula outlined above must be stated as: *tapered result = result (sum of components) – max (withdraw_base – withdraw_start, 0) * withdraw_rate*

If withdraw_end is indicated withdraw_rate is calculated as: *withdraw_rate = result (sum of components) /*

(withdraw_end – withdraw_start)

The policy function SchedCalc

The function SchedCalc is a schedule calculator. Its basic functionality is demonstrated in example 1.

Example 1:

Policy	*Grp/No*	**SL_demo**	*Comment*
SchedCalc		**on**	
base		il_taxableY	
band_upLim	1	5000#y	
band_rate	1	0	income below 5,000 is exempted
band_upLim	2	50000#y	income between 5,000 and 50,000
band_rate	2	0.25	is taxed with a 25% rate
band_rate	3	0.5	income above 50,000 is taxed with a 50% rate
output_var		tin_s	
TAX_UNIT		tu_individual_sl	

In the example the income defined by the incomelist il_taxableY is divided into three bands, where the income band from annually 0 to 5,000 Euro (band_upLim, Grp/No 1) is taxed with a rate of 0% (band_rate, Grp/No 1), the income band from 5,000 to 50,000 Euro (band_upLim, GrpNo 2) with a rate of 25% (band2_rate, GrpNo 2) and all income above 50,000 Euro per year with a rate of 50% (band_rate, GrpNo 3).

An income of 60,000 Euro per year would for example lead to a tax of annually 16,250 Euro (5,000*0%+(50,000-5,000)*25%+(60,000-50,000)*50%) and an income of 25,000 to a tax of 5,000 (5,000*0%+(25,000-5,000)*25%). You may have noticed that there is no explicit definition of the lower limit of the first band. This is not necessary, because it is by default zero. To change this default the parameter band_lowLim, Grp/No 1 can be used. There is also no explicit definition of the upper limit of the third band, as by default the upper limit of the last band is infinite (actually 999,999,999.99). In fact the same schedule could be constructed by skipping the 0% band. Example 2 demonstrates how.

Example 2:

Policy	*Grp/No*	**SL_demo**	*Comment*
SchedCalc		**on**	
base		il_taxableY	
band_lowLim	1	5000#y	income below 5,000 is exempted
band_rate	1	0.25	income between 5,000 and 50,000
band_lowLim	2	50000#y	is taxed with a 25% rate
band_rate	2	0.5	income above 50,000 is taxed with a 50% rate
output_var		tin_s	
TAX_UNIT		tu_individual_sl	

From a technical point of view it is possible to combine lower and upper band limits, but if so this should be done with care, usually it is confusing.

Example 3 shows another option to use the function SchedCalc.

Instead of applying rates on the bands, fixed amounts are used.

Example 3:

Policy	*Grp/No*	**SL_demo**	*Comment*
SchedCalc		**on**	
base		il_taxableY	
band_lowLim	1	5000#y	income below 5,000 is exempted
band_amount	1	500#y	income between 5,000 and 50,000
band_lowLim	2	50000#y	is taxed by a fixed amount of 500
band_amount	2	1000#y	income above 50,000 is taxed by a fixed amount of 1,000
output_var		tin_s	
TAX_UNIT		tu_individual_sl	

For the income band from 5,000 (band_lowLim, Grp/No 1) to 50,000 (band_lowLim, GrpNo 2) an amount

of 500 (band_amount, Grp/No 1) is due and for all

income above 50,000 an amount of 1,000 (band_amount, GrpNo 2) is added. An income of 60,000 would for example lead to a tax of 1,500 and an income of 25,000 to a tax of 500. Obviously it is not possible to define as well a rate as an amount for a single band. However, technically it is possible to mix rates and amounts for different bands (e.g. define a rate for band one and an amount for band two). Yet as it seems rather unlikely that this is necessary, the programme issues a warning in such cases, assuming that this was done by mistake.

The optional parameter quotient is relevant for joint taxation: for couples the income of both partners is added and (in the simplest case) divided by two, then the schedule is applied, to afterwards multiply the resulting tax by two. With progressive tax-schedules this procedure is of advantage for couples where one partner has low (or no) and the other high income, as the average income falls into lower tax bands. Example 4 demonstrates a simple joint taxation schedule.

Example 4:

Policy	*Grp/No*	**SL_demo**	*Comment*
SchedCalc		**on**	
base		il_taxableY	
band_lowLim	1	5000#y	income below 5,000 is exempted
band_rate	1	0.25	income between 5,000 and 50,000
band_lowLim	2	50000#y	is taxed with a 25% rate
band_rate	2	0.5	income above 50,000 is taxed with a 50% rate
quotient		2	couple's income is divided by 2, resulting tax is multiplied by 2
output_var		tin_s	
TAX_UNIT		tu_individual_sl	

If the income of 60,000 is earned by a couple where one partner earns 50,000 and the other 10,000 the resulting tax is 12,500 (50,000 + 10,000 = 60,000; 60,000 / 2 = 30,000; (30,000 – 5,000) * 25% = 6,250; 6,250 * 2 = 12,500) when a quotient of 2 is applied, compared to 16,250 in example 1 (note that the parameter TAX_UNIT is set to couple_sl in example4).

For the sake of completeness, example 5 illustrates how to use the optional and rarely used parameter simple_prog. If this parameter is used, the same rate/amount is applied on all income. The respective rate/amount is the one of the highest band the income falls into.

Example 4:

Policy	*Grp/No*	**SL_demo**	*Comment*
SchedCalc		**on**	
base		il_taxableY	
band_lowLim	1	5000#y	persons with income below 5,000 are exempted
band_rate	1	0.25	persons with income between 5,000 and 50,000
band_lowLim	2	50000#y	are taxed at a rate of 25% on the whole income
band_rate	2	0.5	persons with income above 50,000

simple_prog		yes	are taxed at a rate of 50% on the whole income
output_var		tin_s	
TAX_UNIT		tu_individual_sl	

To use again the two examples with an income of 60,000 respectively 25,000: in the 60,000 case the tax amounts to 30,000 (60,000 * 50%), because the highest band reached with this income is band two with a rate of 50%.

Accordingly, in the 25,000 case the tax is 6,250 (25,000 * 25%).

SchedCalc

provides two further optional parameters. The first, baseThreshold, is a threshold for the base income. If the base income is below this threshold the result(ing tax) is set to zero. The second, roundBase, allows rounding the base income. By setting the parameter to 1 the base income is rounded to whole numbers (e.g. 123.123 to 123 and 789.789 to 790). By setting it to 1,000 the base income is rounded to whole thousands (e.g. 123,123.123 to 123,000 and 789,789.789 to 790,000). By setting it to 0.1 the base income is rounded to have one decimal place (e.g. 123.123 to 123.1 and 789.789 to 789.8).

The policy function Allocate

As outlined in section EUROMOD Functions - Parameter values and the assessment unit, the result of a function is in first instance assigned to the head of the assessment unit. This behaviour may sometimes be unwanted. As an example assume a family benefit, which is in reality assigned to the mother. More often than not the father earns more than the mother and therefore, in the model, he is the head of the assessment unit to whom the benefit is assigned initially. The function Allocate

allows "correcting" this assignment by providing possibilities to

reallocate amounts between members of assessment units. Example 1 shows the simplest application of the function.

Example 1:

Policy	**SL_demo**	*Comment*
Allocate	**on**	
share	bfa_s	
output_var	bfa_s	
TAX_UNIT	sben_family_sl	

In the example the variable *bfa_s* (b=benefit, fa=family, s=simulated) indicated by the parameter *share* is reallocated. The function Allocate accomplishes this by firstly building the sum of the variable over all members of the assessment unit, to afterwards “share”

this sum between assessment unit members with respect to the rules defined by the respective parameter settings. In the example no special rules are defined, therefore the default applies, which is sharing between all members of the assessment unit.

To make the

example more realistic example 2 extends example 1 by the parameter *share_between*, which is set to *!{IsDepChild}*. That means that sharing now involves only adult members of the assessment unit, i.e. children do not get a share (where being a child/adult is determined by the assessment unit’s child definition). In most cases this would mean, that a family benefit, which was assigned in first instance to the father is shared between both parents.

Example 2:

Policy	**SL_demo**	*Comment*
Allocate	**on**	
share	bfa_s	
share_between	!{IsDepChild}	
output_var	bfa_s	
TAX_UNIT	sben_family_sl	

Example 3 illustrates a more complex

reallocation. Tax credits as defined by incomelist *il_taxcredits* (parameter *share*) are shared between members of the assessment unit in proportion to their taxable income as defined by incomelist *il_taxableY* (parameter *share_prop*). The result is

written to the variable *tintc_s* (t=tax, in=income, tc=tax credit) as defined by parameter *output_var*. The parameter *share_equ_ifzero* handles the case that no member of the assessment unit has any taxable income.

If the parameter is set to *yes* equal

sharing takes place, if it is set to *no* or omitted an error message is issued for assessment units without taxable income. Note that no error message is issued if there is nothing to share, i.e.

if there are no tax credits for any member of the assessment unit.

Example 3:

Policy	**SL_demo**	*Comment*
Allocate	**on**	
share	il_taxcredits	
share_prop	il_taxableY	
share_equ_ifzero	yes	
output_var	tintc_s	
TAX_UNIT	sben_family_sl	

Example 4 illustrates that rather

complex reallocations are possible if the full capacity of the parmeter *share_between* is utilised. *share_between* is a condition parameter, i.e. follows the same syntax as the parameter elig_cond

of function elig. The parameter *share_all_ifnoelig* handles the case that no assessment unit member fulfils the conditions set in parameter *share_between*.

If set to *yes* or omitted sharing

takes place among all members of the assessment unit, if set to *no* the output variable is set to zero.

Note, that special care must be taken to avoid “vanishing” benefits or taxes if *share* and *output_var* refer to the same variable and *share_all_ifnoelig* is set to *no*.

Example 4:

Policy	**SL_demo**	*Comment*
Allocate	**on**	
share	sin01_s	
share_between	{dgn=0} & !{IsDepChild} & {IsInEductaion}	
share_all_ifnoelig	no	
output_var	sin02_s	
TAX_UNIT	household_sl	

Finally note, that Allocate always operates on

all members of the assessment unit. That means that the parameter who_must_be_elig is not applicable.

The phase-out function Allocate_F210

Allocate underwent several changes which

also changed the default behaviour of the function. As the former version is still used in some countries' parameter files, a phase-out function Allocate_F210 was introduced, which temporarily maintains the old behaviour. Allocate_F210 will be removed as

soon as there are no applications left in any parameter files. Meanwhile a warning is issued with respect to its phase-out nature.

There are the following differences between Allocate and Allocate_F210:

- With Allocate_F210, if the amount to be shared is defined by a

  variable, it is not compulsory to indicate an output variable (by parameters output_var or output_add_var).

If no output variable is defined the variable to share is also the variable where the function result is written to.

- Allocate_F210 provides a parameter adults_only. If set to yes reallocation involves only adult members of the assessment unit. Note that the default setting of the parameter is yes.
- With Allocate_F210 an error message is issued if the parameter share_all_ifnoelig is set to no and there are no “eligible” persons with respect to the condition defined by share_between. The default setting of the parameter is no.[1]

Hint: The

default behaviour of Allocate_F210 almost can be

obtained by Allocate with the following parameter

settings: output_var set to the same variable as share; share_between set to !{IsDepChild}; share_all_ifnoelig set to no. There is still an important difference:

if there are no adults in the assessment unit, Allocate

sets the output variable to zero while Allocate_F210

shares between all members of the assessment unit, i.e. including children.

[1] If there are no "eligible" persons with respect to *share_between* and *share_all_ifnoelig* is set to *yes* as well as *adults only* sharing takes place among all adults. If there are no adults, the head gets all.

The policy functions Min and Max

The

functions Min and Max calculate the maximum respectively the minimum of a number of values. In (the not very realistic) example below the minimum of employment income (yem), "some income" (defined by the incomelist il_someinc) and a monthly amount of 100 is calculated.

Example1:

Policy	**SL_demo**	*Comment*
Min	**on**	
val	yem	
val	il_someinc	
val	100#m	
output_var	stm01_s	
TAX_UNIT	individual_sl	

Please note, that formula parameters provide the operators <min> and <max>, which may allow a more efficient implementation of minima and maxima.

The system function Uprate

The function Uprate is usually

used in the special policy Uprate_cc

(though it could in principle be used in any policy). It allows for the uprating of monetary dataset variables to the price level of a policy year.

Example 1 shows a typical application of Uprate.

Example 1:

Policy	**SL_2007**	**SL_2008**	**SL_2009**	*Comment*
Uprate	**off**	**on**	**on**	
dataset	n/a	sl_2008_a1	sl_2008_a1	
def_factor	n/a	1.02	1.03	default uprating factor
yem	n/a	1.015	1.025	employment income
yse	n/a	1.025	1.045	self-employment income
Uprate	**off**	**on**	**on**	
dataset	n/a	n/a	sl_2009_a1	
def_factor	n/a	n/a	1.01	default uprating factor
yem	n/a	n/a	1.005	employment income
yse	n/a	n/a	1.015	self-employment income

In the example the dataset sl_2008_a1 (with monetary values referring to the year 2007) does not need any uprating if used with the 2007 system, therefore the first function is switched off for this system. For the other two systems an explicit uprating factor for employment income (yem)

and self-employment income (yse)

is defined. Employment income is uprated by 1.5% for the policy year 2008 and by 2.5% for 2009, self-employment income is uprated by 2.5% for 2008 and by 4.5% for 2009. All other monetary variables in the dataset are uprated by 2% if used with the system SL_2008

and by 3% if used with the system SL_2009. This is

accomplished by the parameter def_factor,

which refers to all monetary variables for which no explicit factor is defined.

The dataset sl_2009_a1 (with monetary values

referring to the year 2008) is not intended to be used with the system SL_2007 and does not need any uprating if used with the system SL_2008, therefore the second function is

switched off for these two system. For the system SL_2009

again an explicit uprating factor for employment income, 5%, and self-employment income, 15%, is defined. All other monetary variables in the dataset are uprated by 1% (def_factor).

Finally, there is no uprating function for the dataset sl_2010_a1, as it cannot be used with the systems SL_2007 and SL_2008, and no uprating is necessary for the system SL_2009.

The function Uprate provides two

parameters intended to enhance transparency, which are presented in Example 2.

Example 2:

Policy	*Grp/No*	**SL_demo**	*Comment*
Uprate		**on**	
name		uprate_sl_demo_v2	
def_factor		1.025	default uprating factor
factor_name	1	cpi	define consumer price index as named factor cpi
factor_value	1	1.02	
yem		cpi	apply named factor cpi on employment income
yse		cpi	apply named factor cpi on self-employment income

The parameters factor_name and factor_value allow for the definition of a "named" factor. In the example the consumer price index is labelled as cpi

by parameter factor_name, Grp/No 1, set to 1.02 by

parameter factor_value, Grp/No 1, and applied on

employment income (yem)

and self-employment income (yse).

Another couple of parameters of the function Uprate allow for "conditional uprating", this is illustrated by example 3.

Example 3:

Policy	*Grp/No*	**SL_demo**	*Comment*
Uprate		**on**	
dataset		sl_demo_a1	
def_factor		1.025	default uprating factor

yem	1	1.02	employment income …
Factor_Condition	1	{drgn1=1}	… in region 1
yem	2	1.025	
Factor_Condition	2	{drgn1=3}	… in region 2
yem	3	1.03	
Factor_Condition	3	{drgn1=3}	… in region 3

In the example employment income (yem) is uprated by three different factors, dependent on region. For the region number one, defined by setting the parameter Factor_Condition; Grp/No 1 to {drgn1=1} (d=demographic, rg=region, n1=nuts level 1), a factor of 1.02 is defined. Analogously, employment income in region number two is uprated by 2.5% (Factor_Condition, Grp/No 2 set to {drgn1=2} and yem, Grp/No 2 set to 1.025) and in region number three by 3% (Factor_Condition, Grp/No 3 set to {drgn1=3} and yem, Grp/No 3 set to 1.03).

Another, not unlikely use of

conditions in the uprate function may however be problematic and therefore exemplified here. Assume, that the variable for old age pension (*poa*) is uprated dependent on its level, i.e. if it amounts up to 1,000 the uprating factor is 1.01, while if it is higher the factor is 1.02. Now assume a person with original *poa*=1,000. Thus the first condition applies and *poa* is uprated to 1,010. Consequently the second condition applies too, as *poa* now exceeds 1,000, with a resulting *poa* of 1,030.2.[1] This is most likely not what was intended, instead one would assume that the conditions refer to the original value of *poa*. However, the programme has no "intuitive knowledge" about the content of the condition and does not take care about such self-references.[2] For the example presented here, a solution may be to use *func_SchedCalc* for uprating instead. Another possibility is to generate a copy of the original *poa* to be used in the conditions.

Furthermore,

some further parameters of the function Uprate allow for consistent uprating of “aggregate variables”, i.e. variables that are composed of other variables. This is illustrated in example 4.

Example 4:

Policy	*Grp/No*	**SL_demo**	*Comment*
Uprate		**on**	

dataset		sl_demo_a1	
def_factor		1.025	default uprating factor
yempj		1.03	employment income in permanent job
yemtj		1.02	employment income in temporary job
aggvar_name	1	yem	employment income is composed of …
aggvar_part	1	yempj	… income in permanent job
aggvar_part	1	yemtj	… income in temporary job
aggvar_tolerance	1	aggvar_tolerance	

In the example employment income (yem) is composed of employment income in permanent jobs (yempj)

and employment income in temporary jobs (yemtj). To make sure for consistent uprating it is not advisable to define a separate uprating factor for the aggregate variable and its components (e.g. 1.025 for yem,

1.03 for yempj and 1.02

for yemtj), as this most

likely results in yempj+yemtj≠yem. The

approach in the example tells the model that an aggregate variable (parameter aggvar1_name set to yem) is composed of particular other variables (parameters aggvar_part1 and aggvar_part2

set yempj respectively yemtj). The

model than "uprates" the aggregate variable by simply building the sum of the already uprated components. Note, that for this reason it is necessary to define the uprating factors of the component variables before defining the composition of the aggregate variable. In principle it is possible that a component variable is itself an aggregate variable (e.g. yempj = yempjag + yempjbs, where ag

stands for agriculture and for bs

business). In such cases the placing of uprating factors and definitions of compositions should be done with great care. Example 4 shows another parameter, aggvar_tolerance. This

parameter allows for imprecise sums. The model checks for each aggregate variable whether it is, before uprating, actually the sum of its components, with a default tolerance of -0.01 to +0.01. The parameter allows for changing this tolerance, in the example it is even tightened.

Finally, there is an option to uprate a whole group of variables at once. This is illustrated in example 5.

Example 5:

Policy	*Grp/No*	**SL_demo**	*Comment*
Uprate		**on**	
dataset		sl_demo_a1	
def_factor		1.025	default uprating factor
RegExp_Def	1	yem*	uprate all employment variables …
RegExp_Factor	1	cpi	… with cpi
RegExp_Def	2	x1*	uprate expenditure variables of the first COICOP group …
RegExp_Factor	2	cpi	… with cpi
RegExp_Def	3	x[2-9]*	uprate all other expenditure variables …
RegExp_Factor	3	1.3	… with 1.3

In the example, all employment variables are uprated with the cpi rate, while expenditure variables are uprated according to their COICOP group. This is feature is very useful if you have an unknown or very large number of variables that you need to uprate simultaneously, as it allows the user to define the variables to be uprated using a regular expression. Note however that this feature cannot be combined with any of the other advanced features such as aggvar or conditions.

[1] For completeness and to fully understand this, it should also be clarified, that the function works cumulatively if more than one condition applies. That means if condition X with the factor x applies as well as condition Y with the factor y, the resulting factor is x*y.

[2] To be precise, it does not take care about any references between variables uprated in the same function.

The system function SetDefault

The

function SetDefault allows for the setting of

default values for not existent dataset variables, as illustrated by the following example, where the dataset hypo_data is used with the default values defined by SetDefault.

Example 1:

Policy	**SL_demo**	*Comment*
SetDefault	**on**	
dataset	hypo_data	name of dataset
yiy	0	income : investment
yot	0	income : other
yempj	yem	income : permanent job

The usage of default values has the effect that the model does not, as usual, issue an error message if a (used) variable does not exist in the dataset, but first looks whether a default is defined for this variable. In the example, if the variables yiy, yot

or yempj do not exist in the dataset, the model assumes yiy and yot to be

zero for each person, while yempj is set to the value of the variable yem (of course yem must exist in the dataset).

Note that

there is another way to avoid error messages for variables not existent in data. If the option use_CommonDefault of the dataset-system combination is set (see Working with EUROMOD - Configuring datasets), all not existent variables are set to zero. The model still considers and prefers any default value set by the function SetDefault.

Summarising, this gives the following behaviour:

Action if variable does not exist		**function SetDefault**	
		defines no default value	*defines a default value*
use_CommonDefault	*no*	issue error message	set this value
	yes	set to zero, no error message	set this value

The system function DefIL

The

function DefIL allows for defining incomelists (see EUROMOD Basic Concepts - EUROMOD terminology).

In principle the function can be applied in any policy. For reasons of transparency incomelists are however usually defined centrally in the special policy ILDef_cc. This rule may be disregarded if a

particular incomelist is just used temporarily in one special policy. Anyway, an incomelist, once defined, is available for all subsequent functions and policies. The examples below illustrate different possibilities to define incomelists.

Example 1:

Policy	**SL_demo**	*Comment*
DefIL	**on**	**taxable income**
name	il_taxableY	
yem	+	employment income
yse	+	plus self-employment income
tscee_s	-	minus employee insurance contributions
tscse_s	-	minus self-employed insurance contributions
DefIL	**on**	**extended taxable income**
name	il_taxableY_ext	
il_taxableY	+	taxable income
poa	+	plus old age pension
...	**...**	**...**
...	...	...
DefIL	**on**	**disposable income**
name	ils_dispy	
ils_origy	+	original income
ils_ben	+	plus benefits
ils_tax	-	minus taxes
ils_sicee	-	minus employee insurance contributions
ils_sicse	-	minus self-employed insurance contributions

Example 1 shows the most common approaches to define incomelists. The first function illustrates the most basic definition of an incomelist as the sum of several variables. The incomelist specifies taxable income and is named il_taxableY (parameter name).

It consists of two variables, which are added (+), yem (y=income, em=employment) and yse (se=self-employment), and two variables, which are subtracted (–), tscee_s (t=tax, sc=social contribution, ee=employee, _s=simulated) and tscse_s (se=self-employed). The second function

in example 1 illustrates that incomelists can be components of other incomelists. It defines extended taxable income by an incomelist named il_taxableY_ext to be composed of the incomelist il_taxableY and the variable poa (p=pension, oa=old age). The third function in example 1 shows the standard definition of EUROMOD disposable income by the incomelist ils_dispy.

It is composed of five other incomelists, where original income (ils_origy) and benefits (ils_ben) are added, whereas taxes (ils_tax) and social

insurance contributions for employees (ils_sicee)

and self-employed (ils_sicse) are subtracted. Note,

that the names of incomelists always start with il_

or ils_, where ils_

denotes system or standard incomelists. These are incomelists, which must be defined for each country. Also note the ... in example 1. They imply that some definitions are not visible, i.e. the definitions of ils_origy, ils_ben, etc., as it is not possible to use an

incomelist as a component of another incomelist without defining it beforehand.

Moreover, if an incomelist is composed of one or more simulated variables, these variables must be calculated before any use of the incomelist, otherwise the model issues a warning.[1] For example, if the simulated variable bch_s

(b=benefit, ch=child, _s=simulated) is a component of a means-test-incomelist, the respective child benefit must be calculated before the means test.

Example 2:

Policy	**SL_demo**	*Comment*
DefIL	**on**	
name	il_AbstractExample	
yem	+0.5	half of employment income
ils_ben	–2	minus 2*benefits

Example 2 creates a not very realistic incomelist called il_AbstractExample, to illustrate that the components of an incomelist can be fractions or multiples of variables or other incomelists.

It subtracts incomelist ils_ben twice from half of

the variable yem.

[1] The model initialises all simulated variables by a value called VOID, which amounts to 0.0000000000001, and issues an error message if such a VOID-variable is used (except of course as an output variable).

The system functions DefTU and UpdateTU

The possibility of tailoring assessment units for any particular purpose, is one of the big strength of EUROMOD, but also requires careful application and some mental effort. The function DefTU allows for defining

assessment units. In principle the function can be applied in any policy. For reasons of transparency assessment units are however usually defined centrally in the special policy TUDef_cc. This rule may be

disregarded if a particular assessment unit is just used temporarily in one special policy.

Once an

assessment unit is defined it can be used by any subsequent function and policy. It is however important to know, at which point of the model run the values of variables (and derivatively incomelists) are assessed. An example may help to understand this issue and its implications. Imagine a family assessment unit, defining a child as a person with "income" below a certain amount. At the start of the model run this income may not be known, as benefits and taxes are not yet calculated. Therefore, assessing the income at this point would need to leave out any simulated variables. As this is not a good solution, the value of variables is assessed once the assessment unit is first used.[1] Note, however, that from this point

the value is not reassessed! That means, if the income refers to a variable that changes after the first usage of the assessment unit and the assessment unit is reused after this change, everybody defined as child in first instance will still be a child, even if the income condition is not valid anymore. The model applies this behaviour as usually it is confusing if a person is a child at one point in time and an adult at another point in time. This is aggravated if units change, i.e. if persons drop out of their original unit, as they are e.g. not a child anymore. As this behaviour still may be unwanted, the function UpdateTU provides the

possibility to reassess any assessment unit conditions, as will be described below. Let’s however start with more basic issues.

Types and members of assessment units

Example 1

illustrates three very basic uses of the function DefTU

and introduces the parameters type and members.

Example 1:

Policy	**SL_demo**	*Comment*
DefTU	**on**	
name	household_sl	
type	HH	
DefTU	**on**	
name	individual_sl	
type	IND	

DefTU	on	
name	family_sl	
type	SUBGROUP	
members	Partner & OwnChild	

The three definitions of assessment units in example 1 mainly differ by their composition, which is primarily described by the parameter type. There are three possible types:

- HH denotes that all members of the household belong to the same assessment unit.
- IND denotes that each member of the household forms its own assessment unit.
- SUBGROUP denotes that the household may be split into several assessment units of different size. Which household member belongs to which unit primarily depends on the parameter members.

Some example households may illustrate these types.

description	idhh	idperson	idpartner	idmother	idfather	dag	household_sl	individual_sl	family_sl
typical family	1	101	102	0	0	30	A	A	A
	1	102	101	0	0	28	A	B	A
	1	103	0	102	101	3	A	C	A
	1	104	0	102	101	1	A	D	A
typical family	2	201	202	0	0	56	A	A	A
	2	202	201	0	0	55	A	B	A
lone parent	3	301	0	0	0	35	A	A	A
	3	302	0	301	0	6	A	B	A
single	4	401	0	0	0	25	A	A	A
two singles living together	5	501	0	0	0	22	A	A	A
	5	502	0	0	0	23	A	B	B
large family	6	601	602	606	0	48	A	A	A
	6	602	601	0	0	45	A	B	A
	6	603	0	602	601	20	A	C	A
	6	604	0	602	601	15	A	D	A
	6	605	0	602	601	10	A	E	A
	6	606	0	0	0	70	A	F	B

For the assessment unit household_sl, with parameter type set to HH,

all members of all household types belong to one unit, the unit A.[2] In contrast, for the assessment unit individual_sl,

with parameter type set to IND,

all members of all household types belong to a different unit, units A to F.

For the assessment unit family_sl parameter type is set to SUBGROUP, i.e.

the household is potentially split up into several units. With respect to the parameter members such a unit comprises Partners and OwnChildren. To interpret this one needs to know that these relations are conceived in relation to the "head" of the assessment unit, i.e. the parameter members must be read as follows: an assessment unit, as defined by family_sl, consists of the

"head", his "partner" and their "own children".

The paragraphs below will complain the double quoted terms in detail, for now we settle for their intuitive meaning. Household 1 consists of a head, a partner (note variable idpartner) and their two own

children (note variables idmother/idfather), therefore they all belong to one unit A.

Household 2 consists of a couple, i.e. head and partner, they also belong both to unit A. Household 3 consists of a lone parent and her/his child, i.e. head and own child, consequently they also belong both to unit A. Household 4 is a single household, obviously there can be only one unit (A). As this gets boring, household 5 is split into two one-person units A and B. The two singles are neither partners nor children of each other, therefore they form separate units. Finally, household 6 consists of unit A, comprising the head (person 601), her/his partner (person 602, note variable idpartner) and their children (603, 604 and 605, note variables idmother/idfather). The grandparent (606) forms an own unit B, as she/he is neither partner nor child of the head. Actually, with the information at hand, person 601 was arbitrarily defined as (first) head, which leads us to the next paragraph …

Defining the head of an assessment unit

The basic

definition of the head is more or less the only "hard wired" part of the assessment unit definition and reads as follows: the head is the richest member of the unit; if there are two or more equally rich persons, the oldest is the head; if there are two or more equally rich and equally old persons, the person with the lowest idperson is the head.

"Richest" is defined by the variable or incomelist indicated by parameter HeadDefInc, which is set to ils_origy by default. Age is unsurprisingly defined by the variable dag. In fact, if the assessment unit type is SUBGROUP, finding the head has to be repeated until all members of the household are assigned to a unit.

That means, firstly (simplifying) the richest person of the household is found as first head and all persons fulfilling the relations defined by parameter members are assigned to her/his unit. Then, if any household members are not yet assigned, the richest person among them is found as the second head and all not yet assigned persons fulfilling the relations defined by parameter members are assigned to her/his

unit. The last sentence is repeated until all household members are assigned to a unit. Now it becomes clear, why it was arbitrary to declare person 601 as the first head – it contained the implicit assumption, that he/she is richer than anyone else in the household.

Though the

head condition cannot be erased, it still can be further defined by using the parameter ExtHeadCond, which stands for extended

head condition. Example 2 demonstrates its use.

Example 2:

Policy	**SL_demo**	*Comment*
DefTU	**on**	
name	household_sl	
type	HH	
ExtHeadCond	{dgn=0}	
StopIfNoHeadFound	no	

If an extended head condition is defined, only persons fulfilling the condition can be head. That means, example 2 describes a matriarchal society, as only women can be head. Among the women it is still the richest (oldest, …) who is chosen to be head. This condition, however, entails a problem, as the model, for obvious reasons, does not allow for headless households. So what happens to womanless households? Due to the parameter StopIfNoHeadFound (whose default value is no) the model will drop the extended head condition where necessary, i.e. if there is no person (left within the not yet assigned household members) fulfilling the condition, and the head is found by the usual process. If however the StopIfNoHeadFound parameter was set to "yes", the application would issue an error message once it comes across a household without any women and stop its execution.

Note, that

the extended head condition has a default value of !{IsDepChild}. That means that, if the parameter is not explicitly defined, children cannot be head and, if the parameter StopIfNoHeadFound is also set to yes, the model will issue an error message if there are adultless households. To understand the rational of these defaults, consider the following household, a child condition of being younger than 18 and parameter members set to Partner & OwnDepChild.

idperson	**idpartner**	**idmother**	**idfather**	**dag**	**ils_origy**	**ExtHeadCond= !{IsDepChild}**	**ExtHeadCond={1}**
101	102	0	0	44	0	A	B
102	101	0	0	40	0	A	B

103	0	102	101	14	0	A	B
104	0	102	101	17	100	A	A

The 17 years old person 104 is a dependent child, with respect to the condition assumed above. However, he/she is also the "richest" person in this poor household. Without the extended head condition set to !{IsDepChild}, person 104 becomes the first head, without any other persons in her/his unit (he/she has no partner or children). Person 101 becomes the second head with persons 102 (partner) and 103 (dependent child) in his unit. In contrast, with the extended head condition set to !{IsDepChild}, person 104 is out ruled to be head. Instead person 101, being the oldest, becomes the first head with all other household members in his unit, including person 104, as he/she is his child. Hence, the default extended head condition ensures that dependent children do not get separated from their parents.

Disadvantageously,

the approach entails a problem. If the child condition generates adultless households, the model will issue and error message and stop. This can be avoided by setting the parameter StopIfNoHeadFound

to no, putting up with some child heads. If child

heads are unacceptable, one could still overwrite the extended head condition (e.g. setting ExtHeadCond to {1}),

and use the parameter NoChildIfHead, which is explained

in more detail in the next paragraph.

Before

turning to this, note that the default of parameter ExtHeadCond is overwritten in example 2. If this is unwanted, the parameter must be changed to {Default} & {dgn=0}. This will be explained in

more detail in the paragraph next but one.

Defining dependent children

The

previous paragraph described the conditions for determining the head and the next paragraph will describe a couple of conditions defining other "statuses" within the assessment unit. This paragraph picks up the condition for being a dependent child. An own paragraph is devoted to this issue, not only as is one of the most important parts of assessment unit specifications, but also as it is frequently a prerequisite for other conditions (as it is in fact the case for the already discussed extended head condition !{IsDepChild}). Being a dependent child is specified by the parameter DepChildCond. Example 3 illustrates

the use of this parameter.

Example 3:

Policy	**SL_demo**	*Comment*
DefTU	**on**	
name	household_sl	
type	HH	
DepChildCond	{dag<=15} | ({dag<=19} & {IsInEducation})	

In the example all persons up to 15 years are dependent children, as well as persons up to 19 years, if they are in education. Note, that in this example the child definition has no influence on the composition of the assessment unit – as it is a household assessment unit anyway all household members belong to the same unit. A child definition may however still be necessary, if for example certain benefits depend on the number of children. Sometimes it even makes sense to use the parameter DepChildCond with

an individual assessment unit. Anyway, the next example illustrates the use of the child condition for assigning household members to assessment units.

Example 4:

Policy	**SL_demo**	*Comment*
DefTU	**on**	
name	family_sl	
type	SUBGROUP	
members	Partner & OwnDepChild	
DepChildCond	{dag<=15} | ({dag<=19} & {IsInEducation})	

An example household may provide an easier understanding of the definitions in example 4.

idperson	idpartner	idmother	idfather	dag	IsInEducation	ils_origy	IsDepChild	assessment unit
101	102	0	0	44	no	2500	0	A
102	101	0	0	40	no	1200	0	A
103	0	102	101	21	no	1000	0	B
104	0	102	101	19	no	800	0	C
105	0	102	101	17	yes	0	1	A
106	0	102	101	10	yes	0	1	A

Person 101 is the first head of this household, as he is the richest. He forms the assessment unit A together with person 102, his partner, and persons 105

and 106, their dependent children. Persons 103 and 104 are no dependent children with respect to the definition of parameter DepChildCond.

Therefore, they do not belong to their father's assessment unit, but form own units B and C. You may be puzzled by the fact, that in example 1 (for simplicity reasons) no child definition was used. In this example members was set to Partner & OwnChild instead of Partner & Own**Dep**Child. The difference will be explained below.

Before turning

to this two further parameters in context with the determination of dependent children are presented in example 5.

Example 5:

Policy	**SL_demo**	*Comment*
DefTU	**on**	
name	family_sl	
type	SUBGROUP	
members	Partner & OwnDepChild	
DepChildCond	{dag<=15} | ({dag<=19} & {IsInEducation})	
ExtHeadCond	{1}	
NoChildIfHead	yes	
NoChildIfPartner	yes	

Again some example households may illustrate what these parameter settings effect. Note, that the default extended head condition, which usually prevents dependent children from being head, is "switched off" – {1} means that everyone fulfils the condition, i.e.

everyone can potentially be a head.

idhh	idperson	idpartner	idmother	idfather	dag	IsInEducation	ils_origy	IsDepChild	assessment unit
1	101	0	0	0	18	yes	0	~~1~~ 0	A
2	201	202	0	0	21	no	1000	0	A
2	202	201	0	0	19	yes	0	~~1~~ 0	A
3	301	302	0	0	40	no	0	0	B
3	302	301	0	0	39	no	0	0	B
3	303	0	302	301	18	yes	50	~~1~~ 0	A

Household 1 consists of just one person, who needs to be the head, as there is nobody else. However, person 101 is with respect to parameter DepChildCond a dependent child, as she/he is in education and not older than 19. With the parameter NoChildIfHead

set to yes this condition is "overruled",

i.e. being a head predominates being a dependent child. Note, that without providing any further parameters, the model would issue an error message for this household and stop, as the default of parameter ExtHeadCond does not allow for child heads. Another way of preventing the stop was described above (setting parameter StopIfNoHeadFound

to no). This again highlights that modellers have

many options at their disposal, to set parameters as is useful for their purposes, however it also demonstrates the necessity of putting some mental effort into getting them right.

Household 2

consists of a young couple. Person 201 is head, as she/he is the richest.

Person 202 is her/his partner and therefore belongs to her/his unit. Person 202

is in education and not older than 19, i.e. a dependent child with respect to parameter DepChildCond. However, with the parameter NoChildIfPartner set to yes this condition is "overruled", i.e. being a partner predominates being a dependent child. Finally, household 3 consists of a couple without income and their 18-year-old child, who has a tiny income. As there are no preconditions for being head, this tiny income makes the 18-year-old first head, though she/he is a dependent child with respect to parameter DepChildCond. Her/his parents do not belong to her/his unit, as they are neither her/his partner nor her/his children, therefore they form an own unit. If this split up of a family is "good" or not depends on the application. The subject will be further discussed in the paragraph next but one.

Defining "statuses" within the assessment unit

So far we

have used the parameter members, without really

explaining how it works. One can intuitively get the meaning of Partners or OwnChildren belonging to the assessment unit, however a clear definition was missing up until now. In fact, there is a parameter, which explicitly defines who is a partner. Example 6 illustrates its use.

Example 6:

Policy	**SL_demo**	*Comment*
DefTU	**on**	
name	couple1_sl	
type	SUBGROUP	
members	Partner	
DefTU	**on**	
name	couple2_sl	
type	SUBGROUP	
members	Partner	
PartnerCond	{head:idperson=idpartner}	
DefTU	**on**	
name	couple3_sl	
type	SUBGROUP	
members	Partner	
PartnerCond	{Default} & {IsMarried}	

The first assessment unit in example 6, couple1_sl, does not explicitly define who is a partner.

We have however seen, that couples still are assigned to the same unit. This happens, because the parameter PartnerCond has a

default value, which is applied by the model if the parameter is omitted. The first and the second assessment unit, couple1_sl and

couple2_sl, are in fact identically, as the

parameter PartnerCond is set to its default value,

which is {head:idperson=idpartner}. To understand this

condition one needs to know that conditions used in assessment unit definitions allow for two special features (in addition to those provided in general by conditions, see section EUROMOD Functions - The policy function Elig). One of them is the possibility to use the prefixes head: and partner:. They denote, that the subsequent variable refers to the head respectively the partner of the unit. Please note, that the prefixes can only be used with variables, but not with incomelists or queries. Knowing this we can interpret {head:idperson=idpartner}, sloppy speaking, as being a partner means that one’s own partner id is set to the head’s id. Finally, the third assessment unit in example 6, couple3_sl, uses

the second "extra feature" provided by assessment unit conditions, {Default}, which simply denotes the default setting of the respective condition. That means, that the parameter PartnerCond of couple3_sl translates to {head:idperson=idpartner}&{IsMarried}.

In other words, this feature provides the possibility to further define the default condition. In the example, partners are not only identified by their id, but in addition they must be married (hopefully to their partner).

Now we are in the position to list

the possible settings of the parameter members and

to interpret their meaning. The parameter allows for the following "types" of unit members, to be combined by &:

- Partner defined by the parameter PartnerCond
- OwnDepChild defined by the parameter OwnDepCond
- LooseDepChild defined by the parameter LooseDepChildCond
- OwnChild defined by the parameter OwnChildCond
- DepParent defined by the parameter DepParentCond
- DepRelative defined by the parameter DepRelativeCond

All of the ...Cond parameters have default values, which are intended to ease model developer’s life, by being set to plausible values. However, what may be plausible in general can make no sense at all for a special purpose.

Therefore developers need to be aware of these default settings and check if they match their particular requirements.

To

understand the shortly following discussion of each ...Cond parameter’s default value, and even more for changing or extending them, it is necessary to know how the model processes the single conditions. For example, if a condition uses the prefix :head, the head must

be identified beforehand. Similarly, if a condition uses {IsDepChild}, the parameter DepChildCond must be evaluated before.

The model interprets the conditions in the following order:[3]

1. DepChildCond
2. LooseDepChildCond
3. assessing head using amongst others ExtHeadCond
4. PartnerCond
5. OwnChildCond
6. OwnDepChildCond
7. DepParentCond
8. DepRelativeCond
9. LoneParentCond

The order

tries to reflect the most common dependencies by interpreting relatively "independent" conditions first, to allow for their usage in subsequent conditions. Note, that not all of the conditions define a "type" of the parameter members, namely DepChildCond, ExtHeadCond and LoneParentCond don’t. ExtHeadCond specifies the head and therewith also defines a "member type" (as explained above, the head is always member and therefore not explicitly listed by the parameter members). DepChildCond

serves two purposes. Firstly, knowing who is a dependent child is frequently a prerequisite for the other "type" conditions. Secondly, the child status is often requested in "normal" conditions of policy functions, e.g. if it entitles to certain benefits. The latter is also the rational for the parameter LoneParentCond. This raises the

question, how a status defined by one of the ...Cond

parameters can be retrieved, to be used e.g. with the function elig. The answer is, that for each condition a respective query exists, e.g. the query connected to the parameter PartnerCond is called IsPartner, the query connected to the parameter

DepChildCond is called IsDepChild,

etc. See section EUROMOD Functions - Queries for a full list.

Now, let’s turn to the default

values of the single conditions. The following listing explains them in words, for a formal (and therewith possibly more precise) definition see section EUROMOD Functions - Summary of parameters for functions DefTU and UpdateTU.

- **DepChildCond:** If the parameter is not explicitly defined, nobody is a dependent child.

  There is however a default setting that can be assessed with {Default}, whose main purpose is to avoid the split up of families and is explained in more detail below.

- **LooseDepChildCond:** The default definition of a "loose" dependent child describes a person, who is a dependent child with respect to DepChildCond, but has neither mother nor father with respect to the variables idmother or idfather.[4] Loose dependent children are frequently a problem, if they form their own assessment unit, as they are not identified as the child of some adult. Imagine a three years old receiving its own child benefit, as it is the head of its own unit. Therefore the parameter members allows for the explicit assignment of loose dependent children – if the type LooseDepChild is

used, they are assigned to the first head's unit. In fact, the three years old example should be theory, as this is bad dataset definition. However, with a generous DepChildCond (e.g. high age limits) loose

dependent children are still possible.

- **ExtHeadCond:** This parameter and its default setting is described above.
- **PartnerCond:** This parameter and its default setting are described above.
- **OwnChildCond:** The default definition of an own child is being the child of the head or her/his partner (as defined by PartnerCond) with

  respect to the variables idmother or idfather.[4] Note, that a such defined child does not need to be a dependent child (as defined by DepChildCond), which may lead to odd constellations. For illustration, imagine an extended family, where a 70-year-old grandfather with a high pension lives with his 40

  years old son and his family. Setting members to OwnChild may result in splitting the household in a unit with the grandfather and his son and another unit with the daughter in law and the children. More about this below.
- **OwnDepChildCond:** The default definition of an own child is being an own child (as defined by OwnChildCond) and being a dependent child

  (as defined by DepChildCond). In general parameter OwnDepChild is preferable over OwnChild to be used with parameter members.
- **DepParentCond:** The default definition of a dependent parent is being the parent of the head or her/his partner (as defined by PartnerCond)

  with respect to the variables idmother or idfather.[5] Note, that this condition in fact does not define any "dependency". That's because such a dependency can hardly be generalised, but depends on circumstances. That means that, if the parameter members includes the type DepParent, it is nearly always necessary to further define the parameter DepParentCond as {Default}&….

- **DepRelativeCond:** The default definition of a dependent relative is {0}, i.e. not being a dependent relative. That means that, if the parameter members includes the type DepRelative, the parameter DepRelativeCond must define such a

  dependent relative.
- **LoneParentCond:** The default definition of a lone parent is being the parent of at least one dependent child (as defined by DepChildCond)

  with respect to the variables idmother or idfather[4] and not having a partner with respect to variable idpartner.

Avoiding to split up families

This section presents some parameters, which may help to avoid the split up of households into odd units. Let’s start this by explaining the rational for the {Default} of the parameter DepChildCond. As briefly mentioned above, if the parameter is not explicitly defined, nobody is a dependent child. There is however a default setting that can be assessed with {Default},

which translates to !{isparent} & {idpartner=0}.

The first part of this setting !{isparent} prevents

that children get separated from their child-parents and the second part {idpartner=0} prevents that partners get separated from their child-partners. This is illustrated by example 7.

Example 7:

Policy	**SL_demo**	*Comment*
DefTU	**on**	
name	family1_sl	
type	SUBGROUP	
members	Partner & OwnDepChild	
DefTU	**on**	
name	family2_sl	
type	SUBGROUP	
members	Partner & OwnDepChild	
DepChildCond	{dag<20} & {ils_origy=0}	
DefTU	**on**	

name	family3_sl	
type	SUBGROUP	
members	Partner & OwnDepChild	
DepChildCond	{Default} & {dag<20} & {ils_origy=0}	

Again some example households may illustrate what these parameter settings effect.

idhh	idperson	idpartner	idmother	idfather	dag	ils_origy	IsDepChild			assessment unit		
							fam1	fam2	fam3	fam1	fam2	fam3
1	101	102	0	0	37	2500	0	0	0	A	A	A
1	102	101	0	0	35	0	0	0	0	A	A	A
1	103	0	102	101	6	0	0	1	1	B	A	A
1	104	0	102	101	1	0	0	1	1	C	A	A
2	201	202	0	0	45	0	0	0	0	A	A	A
2	202	201	0	0	43	0	0	0	0	A	A	A
2	203	204	202	201	19	2000	0	1	0	B	A	B
2	204	203	0	0	18	1500	0	1	0	B	B	B
3	301	302	0	0	45	2000	0	0	0	A	A	A
3	302	301	0	0	43	1500	0	0	0	A	A	A
3	303	0	302	301	19	0	0	1	0	B	A	B
3	304	0	303	0	0	0	0	1	1	C	B	B

To come straight to the point: the first assessment unit definition in example 7, family1_sl, is nonsense, as it does not

provide a definition of dependent children. The second definition, family2_sl, at first view seems ok, but it involves some problems. Finally, the third definition, family3_sl,

overcomes these problems, whether however the suggested solution is the desired depends on the concrete requirements. Now, let’s investigate these statements.

Household 1 is a typical household: mother father and two small children. family1_sl does not even come up with this common family structure. The children form their own unit as, due to the missing parameter DepChildCond, nobody is a dependent child. family2_sl and family3_sl seem ok, persons 103 and 104 are dependent children and therefore assigned to unit of their parents. Household 2 is less typical: a young couple, having no income, lives with the parents of one partner. In this case all three family definitions show a split that is not obviously odd. With the definitions of family1_sl and family3_sl the young couple forms an own unit. In both cases this is caused by the fact that the 19 years old is not a dependent child and therefore gets split from her/his parents. With the definition of family1_sl this is

caused by the missing dependent child condition, in contrast with the definition of family3_sl the 19 years old is a

dependent child with respect to the age and income conditions, however she/he does not fulfil the condition {idpartner=0}. family2_sl separates the child in law from the rest, as the 19 years old in this case is a dependent child and therefore belongs to the unit of her/his parents. As a result the child in law is left over without any relation entitling her/him to belong to the unit of the parents of her/his partner. Whether any of these splits is ok, depends on requirements. Finally, household 3 comprises a young mother, having no income, who lives with her child and her parents. As with household 1, family1_sl

shows an odd split due to the missing dependent child condition. This time also family2_sl exhibits an odd split, as it separates the

small child from the rest of the family, for the same reasons why the child in law was split from the rest in household 2. The grandchild has no relation entitling it to belong to the unit of her grandparents. Note, that not even including loose dependent children would solve the problem, because the baby is no loose child, as it has a mother. Whether the split with family3_sl is desired depends on requirements, at least it is plausible. The young mother stays together with her child, as the condition !{isparent}

out rules her for being a dependent child herself, but the two get separated from the grandparents.

As

foretelled the splits of family3_sl are "not

odd", but intuitively one would think that the 19 year old with a partner and the young mother, both having no income, are dependent children and therefore should belong to the units of their parents. Example 8 illustrates how to accomplish this by using the parameters AssignDepChOfDependents and AssignPartnerOfDependents, which do what their

names imply.

Example 8:

Policy	**SL_demo**	*Comment*
DefTU	**on**	
name	family4_sl	
type	SUBGROUP	
members	Partner & OwnDepChild	
DepChildCond	{dag<20} & {ils_origy=0}	
AssignDepChOfDependents	yes	
AssignPartnerOfDependents	yes	

The table below shows how this setting effects the three households introduced above.

idhh	idperson	idpartner	idmother	idfather	dag	ils_origy	IsDepChild	assessment unit
1	101	102	0	0	37	2500	0	A
1	102	101	0	0	35	0	0	A
1	103	0	102	101	6	0	1	A
1	104	0	102	101	1	0	1	A
2	201	202	0	0	45	2000	0	A
2	202	201	0	0	43	1500	0	A
2	203	204	202	201	19	0	1	A
2	204	203	0	0	18	0	1	A
3	301	302	0	0	45	2000	0	A
3	302	301	0	0	43	1500	0	A
3	303	0	302	301	0	19	1	A
3	304	0	303	0	0	0	1	A

With the definitions of family4_sl the

incomeless child of household 2 and the young mother of household 3 now belong to the units of their parents. But also the child in law of household 2 belongs to the unit of her/his parents in law, as well as the small child of household 3 belongs to the unit of its grandparents, as they now have relationships (to their partner respectively mother) entitling them to it.

Using conditions which refer to income

Income related conditions are frequently used elements in defining assessment units.

For example, child definitions often contain upper limits on income. However, as elaborated in section EUROMOD Functions - Parameter values and the assessment unit, once a unit bigger than the individual is concerned the level of interpreting variables or incomelists is not necessarily intuitively clear at the outset. Therefore, the following examples intend to exemplify how to get income related conditions right. To understand the examples, please note, that the rules outlined in section EUROMOD Functions - Parameter values and the assessment unit are also valid for the function DefTU.

Example 9 illustrates a child condition, which refers to the child's income.

Example 9:

Policy	*Grp/No*	**SL_demo**	*Comment*
DefTU		**on**	
name		example1_sl	
type		SUBGROUP	
members		Partner & OwnDepChild	
DepChildCond		{dag<20} & {ils_origY#1<200#m}	
#_level	1	individual_sl	

In the example dependent children are persons younger than 20, with own income, as defined by incomelist ils_origY, below

200 Euro monthly. Note, that without specifying that ils_origY

should be assessed on individual level it would be assessed on assessment unit level.

Example 10

illustrates a child condition, which refers to the income of the child's parents.

Example 10:

Policy	*Grp/No*	**SL_demo**	*Comment*
DefTU		**on**	
name		example2_sl	
type		SUBGROUP	
members		Partner & OwnDepChild	
DepChildCond		{dag<20} & {GetParentsIncome#1<=1000#m}	
#_income	1	ils_origy	

In the example dependent children are persons younger than 20, whose parents' (as defined by variables idmother and idfather[6]) income, as defined by incomelist ils_origY, does not

exceed 1,000 Euro monthly. (For the definition of the query GetParentsIncome also see section EUROMOD

Functions - Queries.)

Example 11

shows two dependent parent conditions, which refer to the joint income of a potentially dependent parent and her/his partner.

Example 11:

Policy	*Grp/No*	**SL_demo**	*Comment*
DefTU		**on**	
name		example3_sl	
type		SUBGROUP	
members		Partner & OwnDepChild & DepParent	
DepChildCond		{dag<20}	
DepChildCond		{dag>60} & {GetParentsIncome#1<=1000#m}	
#_income	1	ils_origy	
DefTU		**on**	
name		example4_sl	
type		SUBGROUP	
members		Partner & OwnDepChild & DepParent	
DepChildCond		{dag<20}	
DepChildCond		{dag>60} & {ils_origy#1<=1000#m}	
#_level	1	couple_sl	

By and large the two assessment unit definitions in example 11 do the same. Both define dependent parents as persons aged

older than 60. The joint income, as defined by incomelist ils_origY, of the person and her/his partner may not exceed 1,000 Euro monthly. The only difference is, that the query GetCoupleIncome used in the first version does not allow for an own definition of "partner", but refers to the partner

defined by the variable idpartner (also see section EUROMOD Functions - Queries), while in the second version the assessment unit couple_sl may use the parameter PartnerCond to specify who is a partner.

Example 12

illustrates a dependent parent condition which refers to the potentially dependent parent’s own income as well as her/his partner’s income.

Example 12:

Policy	*Grp/No*	**SL_demo**	*Comment*
DefTU		**on**	
name		example4_sl	
type		SUBGROUP	
members		Partner & OwnDepChild & DepParent	
DepChildCond		{dag<20}	
DepChildCond		{dag>60} & {ils_origy#1<=1000#m} & {GetPartnerIncome#2<=500#m}	
#_level	1	individual_sl	
#_income	1	ils_origy	

In the example dependent parents are

persons aged older than 60, whose own income, as defined by incomelist ils_origY, does not exceed 1,000 Euro monthly. Moreover, the income of their partner (with respect to variable idpartner) may not exceed 500

Euro monthly. (For the definition of the query GetPartnerIncome see section EUROMOD Functions - Queries).

Updating assessment units

As outlined above, household members are assigned to respective units once an assessment unit is first used. This assignment is not changed with subsequent uses, even if circumstances change, i.e. some conditions are not fulfilled anymore or get fulfilled at a later point in the model run. However, the reassessment of the units can be enforced by using the function UpdateTU.

Example 13 intends to illustrate these procedures.

Example 13:

Policy	*Grp/No*	**SL_demo**	*Comment*
DefIL		**on**	**incomelist definitions**
name		il_earns	
yem		+	
yse		+	
DefIL		**on**	
name		il_MeansTest	
il_earns		+	
bed_s		+	
DefTU		**on**	**assessment unit definitions**
name		individual_sl	
type		IND	
DefTU		**on**	
name		family_sl	
type		SUBGROUP	
members		Partner & OwnDepChild	
DepChildCond		{dag<25} & {il_MeansTest#1<=500#m}	
#_level	1	individual_sl	
ArithOp		**on**	**child benefit**
formula		0	initialise variable bed_s
output_var		bed_s	for the means test
TAX_UNIT		individual_sl	
ArithOp		**on**	
formula		50#m * nDepChildrenInTu	
output_var		bch_s	
TAX_UNIT		family_sl	
Elig		**on**	**education benefit**
elig_cond		{IsInEducation} & {dag>17} & {il_earns#1<1500#m}	
#_level	1	family_sl	
TAX_UNIT		individual_sl	
ArithOp		**on**	
who_must_be_elig		one	
formula		600#m	
output_var		bed_s	
TAX_UNIT		individual_sl	
UpdateTU		**toggle**	**assessment unit update**
name		family_sl	
BenCalc		**on**	**social assistance**
comp_cond	1	{IsHead}	
comp_perElig	1	1000#m	
comp_cond	2	{IsPartner}	
comp_perElig	2	500#m	
comp_cond	3	{IsDepChild}	
comp_perElig	3	100#m	
withdraw_base		il_earns	
withdraw_rate		1	
output_var		bsa_s	

TAX_UNIT		family_sl	

Please note, that packing all these definitions and policies into one policy sheet would be very bad programming style, and here is just done for the sake of simplification. Let's very briefly verbalise the essential ongoings in the example, to then research the implications by looking at a sample household.

The relevant assessment unit is family_sl,

consisting of head, partner and own dependent children, where dependent children are defined as being younger than 25 and having no own income above 500 Euro monthly. The relevant income is composed of earnings and an education benefit, received by students older than 17, whose families dispose over earnings of less than 1,500 Euro monthly. Two benefits are calculated using the assessment unit family_sl: firstly, a child benefit

paying monthly 50 Euro per child, and secondly, a social assistance benefit paying 1,000 Euro monthly for the head, 500 Euro for the partner and 100 Euro for each dependent child in the assessment unit. Earnings are totally withdrawn from this benefit. The following table shows the outcomes of these benefits for an exemplary household, once with not updating the units defined by family_sl for its second application and once with updating them. Note, that the switch of the function UpdateTU

is set to toggle. For the example this is to be

understood as it could be either switched on or off, as circumstances require.

idperson	idpartner	idmother	idfather	dag	IsInEduc	il_earns	il_Means Test	il_Means Test upd.	family_sl	family_sl updated	bsa_s not updated	bsa_s updated
101	102	0	0	45	0	1000	1000	1000	A	A	700	600
102	101	0	0	40	0	0	0	0	A	A	0	0
103	0	102	101	18	1	0	0	600	A	B	0	1000
103	0	102	101	10	0	0	0	0	A	A	0	0

The last two columns of the table

show that there is a considerable difference in social assistance depending on whether the units are updated or not before calculating this benefit. This is caused by the 18-year-old student receiving the education benefit, which is part of the means test of the dependent child condition. As the education benefit is however calculated after the first use of the assessment unit family_sl in the child benefit, the 18 year old is initially identified as dependent child and belongs to the unit of her/his parents. Only if the function UpdateTU is used to

update the units, she/he forms her/his own unit for the social assistance benefit.

Excursus: using footnote parameters to change the assessment unit of operands

Section EUROMOD Functions - Footnote parameters for the further specification of operands explains how to change the level of assessment for single operands of a formula or a condition. A special rule must be taken into account when this possibility is used, which is explained here and not already in this section, as the knowledge imparted in the current section is of advantage in understanding the rule. Moreover, though more advanced users definitely should be aware of it, the rule better fits in an excursus, as it is a bit challenging. The rule reads: **the assessment unit of the operand must "contain" the assessment unit of the function**. Consider the following examples to understand this "container rule".

Example 14:

Policy	**SL_demo**	*Comment*
ArithOp	**on**	**workable example**
formula	500#m – yem#1	
#1_level	household_sl	
lowlim	0	
output_var	sin02_s	
TAX_UNIT	family_sl	
ArithOp	**on**	**not workable example**
formula	500#m – yem#1	
#1_level	family_sl	
lowlim	0	
output_var	sin02_s	
TAX_UNIT	household_sl	

The first function is workable (though one could question its sense).

The function’s assessment unit is the family, while the assessment unit of the operand yem is changed to household. Households without

question contain (one or more) families. Thus each family receives 500 Euro minus all employment income received by anyone in the household (containing the respective family). The second function puts things the other way round. The function’s assessment unit is the household, while the assessment unit of the operand yem is the family. What would this mean?

Each household receives 500 Euro minus the employment income of which family within this household? Please note, that a violation of the container rule leads to an error message.[7]

For the

sake of order, a rule needs an exception. The exception for the container rule concerns operands in conditions. They can be changed to any assessment unit.

Example 15 illustrates why.

Example 15:

Policy	*Grp/No*	**SL_demo**	*Comment*
Elig		**on**	
condition		{dag<50} & {yem<1500#m} & {poa#1=0}	
#_level	1	household_sl	
TAX_UNIT		family_sl	

In the example a household member becomes eligible if she/he is younger than 50, the employment income of her/his family does not exceed 1,500 Euro monthly, and nobody within the household she/he lives in receives old age pension. The formulation "a household member becomes eligible" instead of "the household becomes eligible" already provides a hint for the rational of the condition exception. In fact conditions are always evaluated on individual level, any assessment unit level condition just means that all individuals within the assessment unit fulfils the condition. That means, on closer examination, the condition exception is not really an exception, as one always changes the assessment level from individual to some other unit that necessarily must contain the individual as the smallest unit.

[1] A rather special case should be mentioned in this context. If an assessment unit is used with the parameter *#i_level*, assessment unit formation only takes place if the function is carried out for at least one unit. That means if no unit is ‘eligible’ (parameter *who_must_be_elig*) the assessment unit is not formed.

[2] Please note, that the notation A,B,…,F just serves explanation purposes and is not really used by the model.

See section EUROMOD Functions - The system function DefOutput, unitinfo parameters, for the possibility of putting out assessment unit settings.

[3] Note, that the list indicates the order in which the model operates the conditions, but not the order in which the parameters should be defined, i.e. it is for example no problem to define the parameter DepPartnerCond

before the parameter DepChildCond.

[4] Respectively with respect to the variable idparent,

if this variable is used in the dataset instead of idmother and idfather.

[5] Respectively with respect to the variable idparent,

if this variable is used in the dataset instead of idmother and idfather. In this case also a person is a dependent parent, whose partner (with respect to variable idpartner)

is the parent of the head or her/his partner. The latter takes into account, that only one parent can be defined by the variable idparent and assumes that this parent's partner is the other parent.

[6] Respectively with respect to the variable idparent,

if this variable is used in the dataset instead of idmother and idfather.

[7] To complicate things a bit further, individual assessment units are exempted from the container rule as another rule applies: the individual within the unit to be taken into account is the head. With this information the second function of the example would read as follows, if the parameter *#1_level* was set to *individual_sl*: each household receives 500 Euro minus the employment income of the head. To your relief, this exemption is nearly ever applicable and just mentioned as a warning, as such (usually intransparent or even wrong) modelling does not lead to an error message. The reason for the exception is of technical nature, as it allows for the following (sensible) exception with respect to conditions.

The system function DefOutput

The function DefOutput allows for defining

EUROMOD output. Usually one policy, containing one function DefOutput, defines the content of one output file. Though in principle the function could be applied in any policy this may not be advisable for transparency reasons. Example 1 illustrates the different options in defining output.

Example 1:

Policy	**SL_demo**	*Comment*
DefOutput	**on**	
file	example_output_1	name of output file
var	idperson	id of person
il	ils_dispy	disposable income
DefIL1	ils_dispy	variables contained in incomelist disposable income
nDecimals	2	print two decimals
TAX_UNIT	individual_sl	output on individual level

The parameter file indicates the name of the

text file, where the output should be written to, in the example (not very inventive) example_output_1. The extension .txt can

be omitted. Note, that the output path is defined in the run dialog (see Working with EUROMOD - Running EUROMOD). The parameter TAX_UNIT defines whether output should be

on individual level, as is the case in the example, or on another assessment unit's level – what this means will be explained on basis of example 2. The parameter nDecimals defines the number of digits after the

comma, in the example two. That means, 10.1234 will be outputted as 10.12, 10.6789 as 10.68 and 10 as 10.00. The rest of the parameters determine the content of the output file. The parameter var simply

tells that the variable idperson should be printed.

The parameter il tells that the incomelist ils_dispy should be printed, in this case this concerns the value of the incomelist (i.e. the sum of the comprised variables). The parameter DefIL, in contrast, tells that the value

of each entry of the incomelist ils_dispy should be printed.

In example 2 the parameter TAX_UNIT is set to household level. Therefore, output is aggregate on household level (i.e. one row for each household), where rules of aggregation are those defined in section EUROMOD

Functions - Parameter values and the assessment unit. That means employment income (yem) and disposable income (ils_dispy) are household employment respectively disposable income. Similarly the variables comprised in disposable income (parameter DefIL1) are printed on household level. The variable idhh is the same for the whole household, i.e. there is no ambiguity. Whereas, the variable dag (d=demographic,

ag=age) refers to the age of the head of household, as defined by the rules in section EUROMOD Functions - Parameter values and the assessment unit.

Example 2:

Policy	**SL_demo**	*Comment*
DefOutput	**on**	
file	example_output_2	name of output file
var	idhh	household id
var	dag	age of head of household
var	yem	household employment income
il	ils_dispy	household disposable income
DefIL1	ils_dispy	components of disposable income on hh level
TAX_UNIT	household_sl	output on household level

Example 3 illustrates the application of another couple of parameters provided by the function DefOutput. They allow the determination of the "status" of single assessment unit members within the unit.

Example 3:

Policy	*Grp/No*	**SL_demo**	*Comment*
DefOutput		**on**	
file		example_output_3	name of output file
var		idperson	id of person
var		dag	age
var		idpartner	partner’s id
var		idmother	mother’s id
var		idfather	father’s id

il		ils_origy	person’s age
unitinfo_id	1	HeadID	id of head of household
unitinfo_id	1	IsPartner	person is partner of head
unitinfo_id	1	IsDependentChild	person is a dependent child
unitinfo_id	1	IsLoneParent	person is a lone parent
unitinfo_tu	1	household_sl	unit info variables refer to whole household
TAX_UNIT		indivdual_sl	output on individual level

The example may produce the following output (inventing two illustrative households and assuming plausible head, child, partner and lone parent definitions for the assessment unit household_sl).

idperson	idpartner	idmother	idfather	dag	ils_origy	household_sl_HeadID	household_sl_IsPartner	household_sl_IsDepCh
101	102	0	0	40	3000	101	0	0
102	101	0	0	35	0	101	1	0
103	0	102	101	3	0	101	0	1
104	0	102	101	1	0	101	0	1
201	0	0	0	35	2500	201	0	0
202	0	201	0	6	0	201	0	1

The head of household 1 is the 40 years old father, as his original income (ils_origy) is the highest, therefore the variable HeadID is set to his idperson for all assessment unit members. The variable IsPartner is set to 1 for the 35 years old mother, as she is the head’s partner denoted by the variable idpartner.

For the two children, aged one and three years, the variable IsDepChild is set to 1. The head of household 2 is the 35 years old mother. The variable IsLoneParent is set to 1 for her, as she lives alone with her six years old child. For the child the variable IsDepChild is set to 1.

Note, that the parameter unitinfo_tu set to household_sl is the relevant assessment unit for determining the variables HeadId, IsPartner, IsDepChild and IsLoneParent, while the parameter TAX_UNIT

is set to individual_sl. Setting the parameter TAX_UNIT to e.g. household_sl would not make much sense, as there would be only one row for each household, not allowing for a proper indication of the assessment unit variables.

Finally, a comment on a specific warning in

context with DefOutput. If a variable not contained

in the variable description file is listed in standard output (any output file named *_std*.txt) the program, as an attempt to keep

standard output clean, issues the warning "Use of non-standard variable 'xxx' in standard output. Consider to use a standard variable instead.".

The system functions DefVar and DefConst

The function DefVar allows for the definition of intermediate variables, as illustrated by the following example.

Example 1:

Policy	**SL_demo**	*Comment*
DefVar	**on**	**define variables for minimum and maximum of child benefits**
mincb	0	
maxcb	0	
ArithOp	**on**	**set value of minimum**
formula	il_earns*20%	minimum for child benefits is 20% of earnings
output_var	mincb	
TAX_UNIT	sben_family_sl	
ArithOp	**on**	**set value of maximum**
formula	il_earns*80%	maximum for child benefits is 80% of earnings
output_var	maxcb	
TAX_UNIT	sben_family_sl	
BenCalc	**on**	**education child benefit**
...	...	
lowlim	mincb	
uplim	maxcb	
output_var	bched_s	
TAX_UNIT	sben_family_sl	
BenCalc	**on**	**child benefit for birth/adoption**
...	...	
lowlim	mincb	
uplim	maxcb	
output_var	bchba_s	
TAX_UNIT	sben_family_sl	
BenCalc	**on**	**large family child benefit**
...	...	
lowlim	mincb	
uplim	maxcb	
output_var	bchlg_s	
TAX_UNIT	sben_family_sl	

In the example three child benefits are constrained by the same minimum (20% of earnings) and maximum (80% of earnings). For this purpose respective variables for the miminum (mincb) and maximum (maxcb) are generated with

DefVar.

Note that the variables are initially set to zero and the function *ArithOp* is used to define their value. The reason for this approach is that *DefVar* is designed to define variables and possibly initialise them with constants (as e.g. zero), but not to fill them with person or household specific values (note that the function has no *TAX_UNIT*).

In principle variables created by DefVar can be used in the same way as variables defined in the variable description file (see Working with EUROMOD - Administration of EUROMOD variables). However, good modelling practise requires that they are only used as intermediate variables, i.e. storing and outputting of major results is reserved to variables described in the variable description file. In contrast to these "regular" variables there is no naming convention for intermediate variables and it is left to the developer to use "telling" names, optimally something that informs about the purpose of the variable.

The following example shows the usage of the parameter var_monetary. The variable age_plus10 is defined as a non-monetary variable and respectively set to age plus 10.

Note that, if var_monetary is not defined, variables (and constants) are considered monetary, except if they are initiallised with a rate (e.g. *0.03#mr*), in which case they are considered non-monetary.[1]

Example 2:

Policy	*Grp/No*	**SL_demo**	*Comment*
DefVar		**on**	**define a non-monetary variable**

age_plus10	1	0	
var_monetary	1	no	
ArithOp		**on**	**set the variable to age plus 10**
formula		dag+10	
output_var		age_plus10	
TAX_UNIT		individual_sl	
BenCalc		**on**	**some usage of the above defined age variable**
...		...	

The function DefConst allows for the definition of constants: it is common practise, that tax-benefit systems use certain "benchmarks" in several policies. For example, the level of a minimum wage may not only determine the minimum wage itself, but also be used as a benchmark in other policies. Example 3 illustrates how to use the function DefConst for such a purpose. Using constants, allows you to specify the value only once and to use the constant in several functions. It furthermore allows you to have a better overview of how monetary values change overtime. It is in this sense similar to footnotes (x_amount), but pooling information even further (i.e. all key policy parameters in the same place).

Example 3:

Policy	**SL_demo**	*Comment*
DefConst	**on**	
$MinWage	1000#m	
Elig	**on**	
elig_cond	{yem < $MinWage}	
TAX_UNIT	individual_sl	
BenCalc	**on**	
who_must_be_elig	one	

There are multiple differences in the way constants (generated by DefConst) and variables (generated by DefVar) can be used. First of all, variables can be monetary or non-monetary. When they are monetary and used in a function

where the TU contains multiple individuals, they will return the sum of values of all TU members. When they are

non-monetary, they will return the value of the head. Constants on the other hand, although they are always named

as monetaqry, they will always return the value of the TU head. This is because their purpose is to be used as a

fixed (or scalar) number. Secondly, if a constant is defined without a *Condition*, then the same value will

be applied to all observations, and this constant can also be used for run conditions (*Run_Cond*). In contrast,

variables and constants defined with conditions, cannot be used in run conditions, as their value could be different

for each individual. Finally, according to the EUROMOD naming conventions, a constant's name should always start with

the $ character (e.g. $MinWage). The designed purpose of conditional constants is to be used for cases where a specific

threshold or amount can be different for different observations in the database. For example in UK, the minimum wage

or the housing benefit could be different in London compared to other areas.

Example 4:

Policy	*Grp/No*	**SL_demo**	*Comment*
DefConst		**on**	
$MinWage		1000#m	
$ChBen		200#m	
$UnivCredit		500#m	
Condition	1	drgn1 = 8	London
$MinWage	1	1200#m	
$ChBen	1	250#m	
$UnivCredit	1	600#m	
Condition	2	drgn1 >10	Rural UK
$MinWage	2	800#m	
$ChBen	2	180#m	
$UnivCredit	2	450#m	

In the above example, the minimum wage, child benefit and universal credit isare different based on the region. Similarly, one

could use any condition (e.g. "dag < 25" for the minimum wage in some years in Greece) to specify a different constant

value for each individual. Each constant can have only one definition without a condition, which will act as the base/default value,

and multiple definitions of the same constant with conditions (also note that each condition group can contain multiple constants).

If there is no default value defined, and there is also no matching condition, then a warning is issued and the constant is given

the value 0. If more than one conditions are true, the values are applied in group order, so the matching condition with the biggest

group number will determine the final constant value.

[1] More precise *initiallised with a rate* translates to *initiallised with something ending with #?r*.

This means that the mechanism may have unexpected results with formulas. For example *300#m * 0.03#mr* is considered (counterintuitive)

as non-monetary and *0.03#mr * 2* is considered monetary.

The special functions Loop and UnitLoop

The looping

functions allow for repeating part (or all) of the tax-benefit calculations. As an example, for calculating marginal tax rates at least part of the policies need to be calculated twice, once for original income and once for marginally increased income. *Loop* allows for such a loop over a group of policies. The loop is carried out until the number of scheduled iterations is reached and/or the break condition is fulfilled. As certain calculations may depend on the current iteration a

variable called *loopcount_loopid* is provided. If for example the identifier (parameter *loop_id*) of the loop for the marginal tax rate calculations is *mtr*, the

variable *loopcount_mtr* will take a value of 1 in the first loop and 2 in the second. This allows for a respective condition to increase income in the second loop.

Moreover, if certain policies within the loop should not be repeated, they can be switched off after the first iteration, by using *ChangeParam* with the run_cond *{loopcount_x>1}*.

UnitLoop is a bit more special, it is carried out for all "entitled units" within the

household. Marginal tax rate calculations again serve well to explain this.

Households frequently do not comprise only one person with (labour)income.

Assume that a household's marginal tax rate should be calculated as the average of the tax rates realised if each of the (labour)incomes is marginally

increased in turn. *UnitLoop* is designed to allow such computations.

"Entitled units" are in this case all persons with (labour)income. As loop *UnitLoop* provides a loop count variable.

This variable takes a value of one for the first "entitled unit" (i.e.

individual with (labour)income), a value of two for the second, and so on. In addition a variable *IsCurElig_loopid* (e.g. *IsCurElig_umtr* if the *loop_id* of the unit loop is *umtr*) is provided. This variable is true for all members of the entitled unit currently processed and false for all members of other units. These variables allow for a respective condition to increase incomes in turn. In fact, technically *UnitLoop* itself does nothing in particular with the entitled unit (except setting *IsCurElig_x* true). It simply runs the household as often over the functions included in the loop as there are entitled small units. It is the developers task to implement "actions", by using the variables provided by the loop. Despite the already mentioned there are three further variables: *nULElig_loopid* indicates the number of entitled small units within the household (and therewith the number of iterations for this household); it is set (to the same value) for all household members. *IsULElig_loopid* indicates whether a specific small unit within the household is entitled; it is set to one for all members of an entitled unit and to zero for all members of a not entitled unit. *IsEligInIter_loopid* indicates in which iteration a specific small unit is entitled; it is set (to the respective iteration number) for all members of the small unit.

The examples below illustrate these variables:

idhh	idperson	idpartner	yem	nULElig_unit	isULElig_unit	isEligInIter_unit
loop_id=unit; elig_unit=individual, elig_unit_cond={yem>0}						
1	101	102	1500	3	1	1
1	102	101	1000	3	1	2
1	103	0	500	3	1	3
1	104	0	0	3	0	VOID
2	201	0	0	0	0	VOID
3	301	302	2000	1	1	1
3	302	301	0	1	0	VOID
loop_id=unit; elig_unit=couple, elig_unit_cond={yem>0}						
1	101	102	1500	2	1	1
1	102	101	1000	2	1	1
1	103	0	500	2	1	2
1	104	0	0	2	0	VOID
2	201	0	0	0	0	VOID
3	301	302	2000	1	1	1
3	302	301	0	1	1	1

The location of the loop functions is not important as they are

custom-treated by the model in the sense of their principle independency of the policy and function order.

A final

note on encapsulated loops: it may be reasonable that Loop encapsulates UnitLoop and both loops run over the same policies. In this case it is important to know how the programme identifies the "inner" and "outer loop". If the first_xxx/last_xxx parameters are used, the loop which is defined first will be the inner loop. If the start_after_xxx/stop_before_xxx parameters are used, the loop which is defined first will be the outer loop.[1] To make things clearer one could also use the first_xxx/last_xxx parameters for the inner loop and the start_after_xxx/stop_before_xxx parameters for the outer loop

(referring to the functions/policies preceding the first respectively following the last function/policy of the loop).

Examples

for the application of the loop functions can be found in section EUROMOD Functions - The special functions Store and Restore.

[1] This is due to the fact that loops are subsequently inserted using the references indicated in the *first_xxx*/*last_xxx start_after_xxx*/*stop_before_xxx* parameters, i.e. ignoring any meanwhile inserted loop functions.

The special functions Store and Restore

Store and

Restore are primarily, though not

exclusively, designed to be used with the loop functions (see EUROMOD Functions - The special functions Loop and UnitLoop). In this context they mainly fufill two tasks. Firstly, they allow to back-up variables and set them back to their initial (or some other previous) value after each iteration and secondly they support storing the results of each iteration.

Example 1

illustrates the back-up functionality in a stylised way. Store is used to back-up the variables yem, yse and all variables contained in the incomelist il_pensions before the loop "abc" starts. The very last function of the loop is a Restore, which sets the variables back to their value at storing time by simply referring to the corresponding Store (via the parameter postfix).

Example 1:

Policy	**SL_demo**	*Comment*
Loop	**on**	
loop_id	abc	
first_func	ExampleStore_loopstart	assuming that policy is called ExampleStore
last_func	ExampleStore_loopend	
num_iterations	10	
Store	**on**	
postfix	bkup	
var	yem	back-up variable yem
var	yse	back-up variable yse
il	il_pensions	back-up all variables contained in incomelist il_pension
...	**on**	
...	...	do something with the stored variables
Restore	**on**	
postfix	bkup	set the stored variables back to their initial value

In reality loops are rarely placed in just one policy but embrace

several policies or even the whole tax-benefit-calculations. In this case incomelists (parameter ilX) help to store groups of

variables, as shown in the example.

Variables stored by Store cannot only be restored by Restore, rather Store produces for each variable stored a back-up variable. This is illustrated in Example 2.

Example 2:

Policy	**SL_demo**	*Comment*
Store	**on**	
postfix	bkup	store variables poa, yem, yse
il	il_wkinc	il_wkinc=yem+yse
var	poa	
...	**on**	
...	...	do something ... possibly change poa, yem, yse
Restore	**on**	
postfix	bkup	set variables poa, yem, yse back to their value at storing time
DefOutput	**on**	
file	example_out	
var	poa	contains possibly changed value of poa
var	poa_bkup	contains value of poa at storing time
var	yem	contains possibly changed value of yem
var	yem_bkup	contains value of yem at storing time
var	yse	contains possibly changed value of yse
var	yse_bkup	contains value of yse at storing time
il	il_wkinc	=yem+yse (i.e. possibly changed value)
il	il_wkinc_bkup	=yem_bkup+yse_bkup (i.e. value at storing time)
TAX_UNIT	individual_sl	

As can be seen from the example (DefOutput), Store produces one new variable for each variable it stores, irrespective if this variable is indicated directly (parameter var) or contained in an incomelist (parameter il). These

variables inherit the features of their source variables, i.e. if the source variable is for example monetary, the copy is as well. Moreover, where the parameter il is

used a copy of the incomelist definition is generated. In the example this is the incomelist il_wkinc_bkup containing the variables yem_bkup and yse_bkup. In general the variables and incomelists produced by Store can be used as any normal variable or

incomelist. There are minor differences, e.g. (for technical reasons) such incomelists cannot be used with the parameter DefIL of *DefOutput*.

You may have asked yourself what

"postfix"

stands for. This will be answered now (even if you have not posed yourself this crucial question). "post" means that the indicated text is added

to the end of the source variables name in determining the storage variable's name. And "fix"

points to a static text, in contrast to the parameter postloop whose usage is illustrated in example 3.

Example 3:

Policy	**SL_demo**	*Comment*
Loop	**on**	
loop_id	turn	
first_func	ExampleStore_loopstart	assuming that policy is called ExampleStore
last_func	ExampleStore_loopend	
num_iterations	3	
...	**on**	
...	...	do something different in each turn of the loop
Store	**on**	
postloop	turn	store the results of each turn of the loop
var	tin_s	store model-calculated income tax
il	il_sic	store model-calculated insurance contrib. (tscee_s+tscse_s)
DefOutput	**on**	
file	example_out	
var	tin_s_turn1	contains value of tin_s after the 1st iteration of the loop
var	tin_s_turn2	contains value of tin_s after the 2nd iteration of the loop
var	tin_s_turn3	contains value of tin_s after the 3rd iteration of the loop
var	tin_s_turn	contains value of tin_s after the last iteration of the loop
var	tscee_s_turn1	contains value of tscee_s after the 1st iteration of the loop
var	tscee_s_turn2	contains value of tscee_s after the 2nd iteration of the loop
var	tscee_s_turn3	contains value of tscee_s after the 3rd iteration of the loop
var	tscee_s_turn	contains value of tscee_s after the last iteration of the loop
var	tscse_s_turn1	contains value of tscse_s after the 1st iteration of the loop
var	tscse_s_turn2	contains value of tscse_s after the 2nd iteration of the loop
var	tscse_s_turn3	contains value of tscse_s after the 3rd iteration of the loop
var	tscse_s_turn	contains value of tscse_s after the last iteration of the loop
il	il_sic_turn1	contains value of il_sic after the 1st iteration of the loop
il	il_sic_turn2	contains value of il_sic after the 2nd iteration of the loop
il	il_sic_turn3	contains value of il_sic after the 3rd iteration of the loop
il	il_sic_turn	contains value of il_sic after the last iteration of the loop

The example shows (DefOutput) that Store produces one new variable per variable per iteration of the loop. Again, where the parameter il is used copies of the incomelist definition are generated: one per incomelist per iteration of the loop. Moreover, in each case one variable/incomelist without an iteration number is generated (tin_s_turn, il_sic_turn, etc.). These contain the most recent value of the variable/incomelist. That means that after the three iterations of the loop are terminated tin_s_turn has the same value as tin_s_turn3. However, if the variable is used by some function, say after the second iteration of the loop, it has the value of tin_s_turn2. As the second application is

rarely helpful (usually one could use tin_s as well), the main purpose of these variables/incomelists without an iteration number is with loops, which are ended by a condition, i.e. the iteration number of the last turn is unknown after the end of the loop.

It is possible for Restore to refer to a Store applying the parameter post_loop. This is illustrated in example 4.

Example 4:

Policy	**SL_demo**	*Comment*
Loop	**on**	
loop_id	turn	
first_func	ExampleStore_loopstart	assuming that policy is called ExampleStore
last_func	ExampleStore_loopend	
num_iterations	3	
Store	**on**	
post_loop	turn	store variable yem before each iteration of the loop
var	yem	
...	**on**	
...	...	do something with yem
Restore	**on**	
postloop	turn	set yem back to its value before the first iteration
iteration	1	

In the example Restore sets the variable yem back to the value it had when Store was carried out in the first iteration of the loop, i.e. in this case its original value. Actually, the example is quite artificial as it is not very meaningful to store yem at the beginning of the loop (yem_turn3 will finally have the value of yem before the 3rd iteration, i.e. the value after the 3rd iteration gets lost), but illustrates the usage of Restore with postloop in a traceable way. In fact using Restore this way is somewhat tricky and should be done with care. If for example Store was carried out – more meaningfully – at the end of the loop, setting yem back to yem_turn1 would mean to reset it to its value after the first iteration and not to its original value. Moreover, if the parameter iteration

is omitted, variables are set back to the value they had when the corresponding Store was carried out most recently (i.e.

to the current value of the variables without iteration number) – a mechanism usually not easily to follow.

If Store refers to a UnitLoop the set of storage variables and

incomelists is in principle the same, however their content needs some further explanation respectively is different.

Example 5:

Policy	**SL_demo**	*Comment*
UnitLoop	**on**	
loop_id	unit	
first_pol	ExampleStoreUL	assuming that policy is called ExampleStoreUL
last_pol	ExampleStoreUL	
elig_unit	individual_sl	
ArithOp	**on**	
formula	1	
output_add_var	stm01_s	
TAX_UNIT	individual_sl	
Store	**on**	
postloop	unit	set yem back to its value before the first iteration
var	stm01_s	

Above example would produce the

following output:

idhh	idperson	stm01_s_unit1	stm01_s_unit2	stm01_s_unit3
1	101	1	2	3
1	102	1	2	3
1	103	1	2	3
2	201	1	1	1
3	301	1	2	2
3	302	1	2	2

In principle the example simply

counts the iterations of the loop and writes the result to the variable stm01_s (ArithOp adds 1 to stm01_s in each iteration). The number of iterations is determined by the number of individuals in the household (parameter elig_unit

set to individual_sl), i.e. three for the first household, one for the second and two for the third. In the first iteration of the unit loop stm01_s takes a value of 1 for everyone, which is stored in stm01_s_unit1. In the second iteration stm01_s takes a value of 2 for everyone, except for the second household – as there is no second person there is no second iteration for this household, so stm01_s keeps its value of 1. This result is stored in stm01_s_unit2. Finally, in the third iteration stm01_s takes a value of 3 for the first household (which includes three individuals) and keeps its value for the other households (which include less than three individuals). This result is stored in stm01_s_unit3.

If UnitLoop would be extended by the parameter elig_unit_cond set to {dag<50} the output would be the following:

idhh	**idperson**	**dag**	**stm01_s_unit1**	**stm01_s_unit2**
1	101	51	1	2
1	102	48	1	2
1	103	20	1	2
2	201	30	1	1
3	301	58	VOID	VOID
3	302	56	VOID	VOID

The number of iterations this time

is determined by the number of "eligible" individuals in the household, i.e.

persons younger than 50. Consequently there are two iterations for the first household, one for the second and none for the third. Note that for the third household the functions embraced by the loop are not carried out, therefore stm01_s stays undefined (VOID). Also note, that stm01_s takes the same value for each

individual within the household (irrespective of the persons age). This illustrates that UnitLoop

does not care about who is eligible within the household, but simply carries the loop out as often as there are eligible units in the household and leaves it to the modeller to do something with the currently eligible unit.

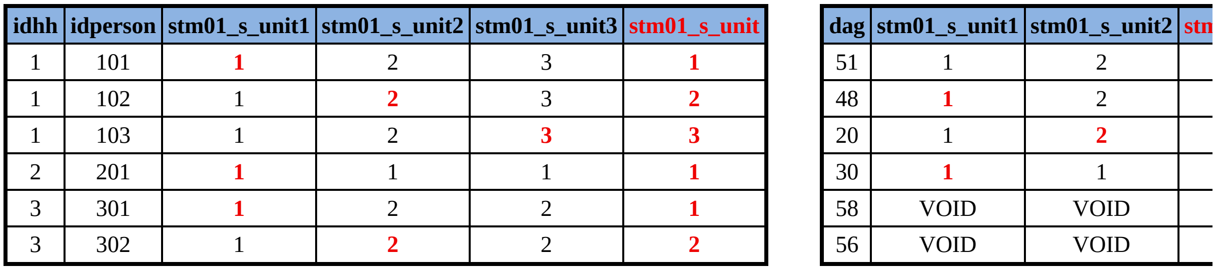
Actually, the output would not

contain "VOIDs" but zeros and, worse, the model would issue a lot of warnings about outputting undefined values. As this is quite likely to happen by using Store referring to a UnitLoop, defoutput provides two useful parameters in this context: suppress_void_message and replace_void_by. The first parameter set to yes obviously avoids the warnings, however to not loose the information, it may make sense to set undefined values to something else than zero, which can be accomplished by the second parameter.

It was not yet mentioned to which

value stm01_s_unit, i.e. the variable without an iteration number, is set. This was kept to the end for two reasons. Firstly, handling these variables/incomelists with Loop is really completely different from its handling with UnitLoop.

Secondly, it may take some effort to understand the mechanism – an effort which is however (hopefully) worthwhile, as it makes output more efficient and, if used properly, also more intuitive. The following tables show the output of the two above examples extended by the variable stm01_s_unit.



idhh	idperson	stm01_s_unit1	stm01_s_unit2	stm01_s_unit3	stm01_s_unit
1	101	**1**	2	3	**1**
1	102	1	**2**	3	**2**
1	103	1	2	**3**	**3**
2	201	**1**	1	1	**1**
3	301	**1**	2	2	**1**
3	302	1	**2**	2	**2**

dag	stm01_s_unit1	stm01_s_unit2	stm
51	1	2	
48	**1**	2	
20	1	**2**	
30	**1**	1	
58	VOID	VOID	
56	VOID	VOID	

The rule leading to these results

is: the variable without an iteration number is set in each iteration to the value of the currently eligible unit.[1] In the left example this means: in the first iteration persons 101, 201 and 301

are eligible, therefore for them stm01_s_unit takes on the value of stm01_s_unit1 and stays VOID for everyone else.

In the second iteration persons 102 and 302 are eligible, therefore for them stm01_s_unit takes on the value of stm01_s_unit2. For everyone else stm01_s_unit keeps its value where it has already one and stays VOID where it has not. Finally, in the third iteration person 103 is eligible, therefore for her/him stm01_s_unit takes on the value of stm01_s_unit3 and keeps its value for everyone else. The story for the right example is similar: in the first iteration persons 102 and 201 are eligible, therefore for them stm01_s_unit takes on the value of stm01_s_unit1 and stays VOID for everyone else.

In the second iteration person 201 is eligible, therefore for her/him stm01_s_unit takes on the value of stm01_s_unit2. For everyone else it stays unchanged, which amongst others means that it stays undefined for persons 101, 301 and 302

as they never were eligible.

These rather technical examples help

to understand the mechanism, but are not very conducive to see what this is good for. A more practical example may contribute to a better understanding.

Example 6:

Policy	**SL_demo**	*Comment*
UnitLoop	**on**	
loop_id	unit	loop over the tax-benefit calculations
start_after_pol	tudef_sl	as often as there are persons with
stop_before_pol	output_std_sl	positive employment income
elig_unit	individual_sl	assuming that this policy is placed at the beginning of the loop
elig_unit_cond	yem>0	(i.e. after tudef_sl)
ArithOp	**on**	
formula	yem_bkup* (1+IsCurElig_unit*0.01)	increase employment income by 1% for each
output_add_var	yem	person with positive employment in turn
TAX_UNIT	individual_sl	assuming that original yem was stored in yem_bkup before the loop
...	**on**	
...	...	...
Store	**on**	
loop_id	unit	store employment income of each iteration
var	yem	assuming that this policy is placed at the end of the loop (i.e. before output_std_sl)

The example produces the following output.

idhh	idperson	yem_bkup	yem_unit1	yem_unit2	yem_unit
1	101	1800	**1818**	1800	**1818**
1	102	1000	1000	**1010**	**1010**
1	103	0	0	0	**0**
2	201	0	VOID	VOID	**VOID**
3	301	2000	**2020**	VOID	**2020**
3	302	0	0	VOID	**VOID**

One can see, that yem_unit shows marginally increased

employment income for each person with positive earnings from employment (VOID for everyone else). Note that yem_unit1 shows the situation in the first iteration of the loop, when yem of

the first person in the household with positive yem is marginally increased (persons 101 and 301) and yem of any other person with positive yem stays unchanged (person 102). yem_unit2 shows the situation in the second iteration of the loop, which is only carried out for households with more than one person with positive yem. Now yem of the second person in the household with positive yem is

marginally increased (person 102) and yem of any other person with positive yem stays unchanged (person 101).

So far this is not a very

interesting finding, what we actually want to know is disposable income in these different situations. Adding parameter il to Store and setting it to ils_dispy would produce the following output:

idhh	idperson	yem_bkup	yem_unit1	yem_unit2	yem_unit	ils_dispy_unit1	ils_dispy_unit2	ils_dispy_unit
1	101	1800	**1818**	1800	**1818**	**1308**	1300	**1308**
1	102	1000	1000	**1010**	**1010**	1050	**1060**	**1060**
1	103	0	0	0	**0**	0	0	**VOID**
2	201	0	VOID	VOID	**VOID**	VOID	VOID	**VOID**
3	301	2000	**2020**	VOID	**2020**	**1500**	VOID	**1500**
3	302	0	0	VOID	**VOID**	0	VOID	**VOID**

This result may still not be

satisfactory, if one assumes that household disposable income is a more telling measure of a person’s means than individual disposable income. The parameter il_level of Store allows to change the assessment level respectively. (There are two further parameters in this context: var_level allows changing the assessment level of a specific variable and level refers to all variables and incomelists of the respective Store.) If

il_level is set to household_sl the output would change as follows:

idhh	idperson	yem_bkup	yem_unit1	yem_unit2	yem_unit	ils_dispy_unit1	ils_dispy_unit2	ils_dispy_unit
1	101	1800	1818	1800	1818	**1308**	**1300**	**2358**
1	102	1000	1000	1010	1010	**1050**	**1060**	**2360**
1	103	0	0	0	0	**0**	**0**	**VOID**
2	201	0	VOID	VOID	VOID	VOID	VOID	**VOID**
3	301	2000	2020	VOID	2020	**1500**	VOID	**1500**
3	302	0	0	VOID	VOID	**0**	VOID	**VOID**

We have now nearly all ingredients

to calculate marginal tax rates on individual level, based on disposable income on household level (yem_bkup, yem_unit, ils_dispy_unit). Still missing is standard disposable income, which may simply be taken from a standard run of the tax-benefit system or – maybe more convenient – calculated by another loop.

The examples hopefully demonstrated

that the storing mechanisms provided by Store allow for output that can be efficiently analysed respectively further processed with statistic tools. However, to properly apply them we need to clarify some details. The following exemplary overview of the

variables/incomelists generated by Store on the one hand summarises what was explained in the text above and on the other hand allows elaborating these details.

Variables/incomelists/constants generated by Store

	postfix=bkup		**postloop=turn (Loop with 2 iterations)**		**postloop=unit (UnitLoop with max. 2 iterations)**		
	variables	**incomelists**	**variables**	**incomelists**	**variables**	**incomelists**	**constants**
var=yem	yem_bkup		yem_turn1 yem_turn2 yem_turn		yem_unit1 yem_unit2		yem_unit
il=il_wkinc[1]	yem_bkup yse_bkup	il_wkinc_bkup[2]	yem_turn1 yem_turn2 yem_turn yse_turn1 yse_turn2 yse_turn	il_wkinc_turn1[3] il_wkinc_turn2[3] il_wkinc_turn[3]	yem_unit1 yem_unit2 yse_unit1 yse_unit2	il_wkinc_unit1[4] il_wkinc_unit2[4]	il_wkinc_unit

[1] il_wkinc = yem + yse

[2] il_wkinc_bkup = yem_bkup + yse_bkup

[3] il_wkinc_turn1 = yem_turn1 + yse_turn1; il_wkinc_turn2 = yem_turn2 + yse_turn2; il_wkinc_turn =

yem_turn + yse_turn [4] il_wkinc_unit1 = yem_unit1 + yse_unit1; il_wkinc_unit2 = yem_unit2 + yse_unit2

While most of the overview should be

self-explaining, we need to clarify why yem_unit and il_wkinc_unit are constants. To do so recall the construction of yem_unit

described above and consider the fact that monetary variables (and incomelists) are assessed on assessment unit level. However, due to its construction yem_unit cannot be assessed on e.g.

household level. For the first household it would than amount to 1818+1010+VOID which does not make any sense. That means the constant-status on the one hand tries to avoid mistakes in this context (but cannot rule them out) and on the other hand highlights the specific contruction. In principle the same is true for incomelists (il_wkinc_unit respectively ils_dispy_unit in the example above). They are generated following the same mechanism described above for yem_unit, i.e. set in each iteration to the value of the currently eligible unit. For the same reasons why yem_unit is a constant rather than a monetary variable il_wkinc_unit and ils_dispy_unit are constant rather than incomelists. A convenient side effect of the constant-status is that a level change via parameters level/var_level/il_level does not generate a (further) inconsistency.

Finally note that Restore cannot refer to a Store applying the parameter post_loop to refer to a unit loop, as this would not be very helpful but potentially most confusing.

[1] More precisely, if the unit is bigger than individual, the variable is set for the currently eligible unit, namely to respective value for the head and to zero for non-heads (opposed to staying undefined for persons within not eligible units).

The special function ChangeParam

The function ChangeParam allows for changing the values of parameters of other functions. Of course usually one would not use a function for this purpose, but change the parameters directly. There are however some cases where using ChangeParam instead makes sense. An evident example is changing parameters from an add-on. In fact, the usage of ChangeParam in add-ons is near to inevitable – how else should parameters of the countries' parameter files be changed from the add-on parameter file? Another case concerns changing parameters from a reform policy without directly changing base policy parameters. Though the reform could as well be implemented in the respective policy sheets (still transparent by using a reform system), in some cases it may be more transparent to summarise the changes in an extra reform policy.

Each parameter change requires two parameters. Firstly, the parameter to change needs to be specified by Param_Id, which allows the indication of a unique identifier for the parameter. Secondly, the new value for the parameter is indicated by the parameter Param_NewVal. The value type of this parameter is determined by the value type of the parameter it changes. That means if the parameter to be changed indicates a yes/no parameter, Param_NewVal also needs to be set either to yes or no.

ChangeParam is independent of the policy/function order defined by the spine. It simply overwrites the former value of the parameter (at read-time) and the program runs as if this value never existed. As a consequence, the function can be located anywhere, without changing its behaviour.

The optional parameter Dataset can be used to restrict the change to specific datasets.
If any such parameter is used (several can be used within one group), the change only takes place if one of them matches the dataset of the concerned run.
The wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g.be_20 * _a ?).

Note that, while the EM2 executable allows for also using ChangeParam to change switches of policies or functions, the EM3 executable provides the function ChangeSwitch for this task.
Also note that changing parameters at run-time (as it was possible with reservations with the EM2 executable by using parameter Param_CondVal) is

not possible with the EM3 executable.

The special function ChangeSwitch

This function is only available with EM3.

The function ChangeSwitch allows for changing the switches of policies or functions. Similar to the function ChangeParam, ChangeSwitch is mainly used in add-ons. (In fact, with the EM2 executable ChangeParam is also used to change switches.)

Different to ChangeParam, ChangeSwitch allows for a Run_Cond, which means, whether the switch-change is carried out or not, may be dependent on run-time conditions.

Note that (in contrast to changing switches with the EM2 executable) the original switch may be off. That means it is not necessary to use toggle anymore.

Also see EUROMOD Functions – Identifiers and the placeholders =cc= and =sys=.

The special function Totals

Totals

allows for the calculation of aggregates (i.e. sums, means, etc.) of variables or incomelists over the whole population (represented by the dataset) or a selected subgroup. The function does not aim to provide a statistical package.

Rather, the aggregates are intended to serve the calculations, as illustrated by example 1. It demonstrates an increase in Simpleland's child benefit, which is financed by a respective raise in income tax. Budget neutrality is achieved by gradually increasing income tax until the higher expenses for the child benefit are covered, which is implemented by means of a loop and Totals to check whether the budget is already balanced.

Example 1: to be revised (use of ChangeParam with old identifiers)

Policy	**SL_demo**	*Comment*
Loop	**on**	**loop function could be placed wherever desired**
loopid	budgneut	
first_pol	sic_sl	
last_pol	cb_reform_sl	assumes that the present policy is called cb_reform_sl
breakcond	{$totnew_ils_dispy<$totold_ils_dispy}	exit condition: tax increase covers raised expenditure
Totals	**on**	**compute total disposable income before the reform**
run_cond	{loopcount_budgneut=1}	
varname_sum	$totold	
agg_il	ils_dispy	
TAX_UNIT	individual_sl	
Totals	**on**	**compute total disp. inc. after each increase of the tax rate**
varname_sum	$totnew	
agg_il	ils_dispy	
TAX_UNIT	individual_sl	
ChangeParam	**on**	
param1_id	sben_cb_sl_#5	assuming the parameter is placed in row 5
param1_condval	500#m	increase child benefit amount (in the 2nd iteration)
param2_id	it_sl_#6	assuming the parameter is placed in row 6
param2_condval	0.2+loopcount_budgneut*0.1%	increasing the tax rate by 0.1% in each loop

Note that variables generated by Totals can be used as any "real"

variables. They take the same value for all persons in the sample, even if the aggregate is built for a sub-sample.

The special functions DropUnit and KeepUnit

DropUnit

and KeepUnit allow for dropping individuals,

families or whole households with certain characteristics from the

calculations. For DropUnit

the parameter drop_cond

indicates a condition determining who is to be dropped, while for KeepUnit the parameter keep_cond describes who is to be kept, i.e.

those who do not fulfil the condition are dropped. The following examples show some applications.

Example 1:

Policy	**SL_demo**	*Comment*
DropUnit	**on**	
drop_cond	{dag<15}	drop all children younger than 15
TAX_UNIT	individual_sl	

Example 2:

Policy	**SL_demo**	*Comment*
DropUnit	**on**	
drop_cond	{poa>0}	all households receiving pension are dropped
TAX_UNIT	household_sl	

Example 3:

Policy	**SL_demo**	*Comment*
KeepUnit	**on**	
keep_cond	{yem>0}	only keep couples where both partners have
keep_cond_who	all	positive employment income
TAX_UNIT	couple_sl	

Example 4:

Policy	*Grp/No*	**SL_demo**	*Comment*
KeepUnit		**on**	
keep_cond		{il_earns>0}#1 & {il_pension=0}#1 & {dag>14} & {dag<66}	keep household if there is at least one person
#_level	1	individual_sl	with positive earnings, not receiving pensions
keep_cond_who		one	and aged between 15 and 65
TAX_UNIT		household_sl	

In example 1 all children younger than 15 are

dropped. Example 2 drops whole households (TAX_UNIT=household_sl), namely those in receipt of pension. In example 3 only singles with positive employment income as well as couples where both partners (keep_cond_who=all) have positive employment income are kept. Note that only the couple (single) is kept, but not their children or other relatives. Finally, example 4 shows a bit more complex use and illustrates the application of different levels of assessment. The keep condition defines persons aged between 15 and 65, who have positive earnings and do not receive pensions. As a default (and as usual) the incomelists il_earns and il_pension would be assessed on TAX_UNIT level, therefore it is necessary to use the level parameter to only take individual incomes into account. This is not necessary for the personal variable dag, which is anyway (also as usual) assessed on individual level. The parameter keep_cond_who tells that at least one person must fulfil the condition for the household to be kept (it could in fact be omitted as one is the default and is only there to make it explicit).

Usually the

dropping of units will take place at the very beginning of the spine, i.e.

before the respective tax-benefit calculations. However, in principle it can be done anytime, as the "time" of dropping the units depends on when the function appears in the policy/module order. Consequently all assessment units built so far are deleted, as the program cannot rely that the persons forming these constellations still exist.

The special function ILVarOp

ILVarOp

allows for operations on the content, i.e. the variables of an incomelist. The following examples show some applications of the function, to illustrate its purposes.

Example 1:

Policy	**SL_demo**	*Comment*
ILVarOp	**on**	
operator_il	il_earns	all variables within the incomelist il_earns
operand	101%	are increased by 1%
operation	mul	parameter could be skipped as 'mul' is default
sel_var	all	parameter could be skipped as 'all' is default

Example 2:

Policy	**SL_demo**	*Comment*
ILVarOp	**on**	
operator_il	il_earns	the variable with the highest value within the
operand	101%	incomelist il_earns is increased by 1%
operation	mul	parameter could be skipped as ‘mul’ is default
sel_var	max	

In the examples variables of the incomelist il_earns are increased by 1%. While in the

former example all variables of the incomelist are increased (sel_var=all), in the latter only the "main"

variable, i.e. the variable with the highest value is increased (sel_var=max). Assuming that il_earns consists of yem and yse this means that in example 1 both employment

income and self-employment income are increased while in example 2 yem is increased if the respective

person’s employment income is higher then her/his self-employment income,

otherwise yse is

increased. If yem and yse are equal, the first (with respect to entries

in the incomelist) is increased.[1] As not hard to guess, ILVarOp can be applied this way in marginal tax rate

calculations. Note that ILVarOp does not provide the parameter TAX_UNIT, that means it always operates on

individual level. Also note that ILVarOp operates on variable level, that means there

is no difference between il_abc=yem+yse+poa and il_def=yem+il_ghi, where il_ghi=yse+poa. Finally, note that the parameter operation could have been skipped in both

examples as mul is

default, as well as parameter sel_var could have been skipped in example 1 as all is default.

Example 1:

Policy	**SL_demo**	*Comment*
ILVarOp	**on**	
operator_il	il_earns	the value of the variable yot is added
operand	yot	to the smallest variable within il_earns
operation	add	
sel_var	min	

In example 3 the parameter operation is set to add instead of mul, which means that the operand is added to all/special variables

within il_earns.

In the example the variable yot (for other income) is added to either yem or yse, dependent on which of the two is smaller.

[1] This may need reconsideration if it

causes problems.

The special function RandSeed

RandSeed

sets the starting point for generating a series of pseudorandom numbers. In order to better understand how random numbers work in EUROMOD, you need to have an understanding of how random numbers work in computers in general.

Random numbers in computers are actually not random at all. Windows (as well as other Operating Systems) have a long list of pages with stored "random" numbers. The operating system keeps a pointer and each time you ask for a random number it returns the next in line, from the page it is currently reading. Changing the "seed" essentially means moving the pointer to a specific page and taking all the numbers there in order.

In practice, this means that if you set a specific "randseed" in the beginning of a system, you can be sure that you will get the exact same sequence of random numbers in every run. if you set the randseed with the same parameter in each system, then you can be sure that all systems get the same sequence of random numbers in every run. If you set the randseed with the same parameter every time before getting a random number, then you will always get the same "random" number.

Few more things to keep in mind are:

- Any series of numbers you get after a randseed should be more or less equally distributed between 0 and 1. The more random numbers you get after a single randseed, the better the distribution. If you set the randseed with every random number you ask for then there is no distribution at all – you always get the same number.
- Randseed will definitely need to be part of your solution if you are using random numbers and want the results to be replicable (i.e.

  if you need to get the same series of random numbers every time)
- If you set the same randseed and then give random numbers to the same individuals with the same order, then they will have the same random numbers across systems. If however the number or order of individuals is changed, then there is no way to link random numbers to individuals.

Example 1:

Example 1:

Policy	**SL_rand1**	**SL_rand2**	*Comment*
RandSeed	**on**	**on**	
Seed	42134	42134	
ArithOp	**on**	**on**	
formula	rand	rand	
output_var	stm01_s	stm01_s	
TAX_UNIT	individual_sl	individual_sl	

In the example the variable stm01_s is filled for each individual with a random number between 0 and 1 (both included).

Assuming both systems use the same data (i.e. the same number of individuals in the same order), you can be sure that for each individual variable stm01_s will have the same value in both systems. Note that, if RandSeed would not be used, or if the *Seed* was different, the program would produce different results for the systems sl_rand1 and sl_rand2.

The special function CallProgramme

CallProgramme allows for calling an external application. For example, a statistic programme, like Stata, can be called at the end of the spine to further process the output produced by EUROMOD. Note that the function is limited to the Windows platform.[1]

Example 1 calls Microsoft Excel to open a certain workbook.

Example 1:

Policy	**SL_demo**	*Comment*
CallProgramme	**on**	
Programme	Excel.exe	
Argument	&Output\SomeExcelWorkbook	
RepByEMPath	&	

The parameter *Programme* defines the application to be called as Microsoft Excel. The parameter *Argument* specifies an argument to be passed to Excel: *&Output\SomeExcelWorkbook*, which is the name and path of the workbook that Excel should open (ignoring the *&* for a moment). The parameter *RepByEMPath* allows for using the path of the current EUROMOD installation for the specification of the parameter *Argument*. Assuming EUROMOD is installed at *C:\EuromodFiles\*, the argument passed to Excel is in fact *"C:\EuromodFiles\Output\SomeExcelWorkbook"*.

[1] This restriction is necessary because a c++ programme has, apart from multithreading, only one possibility to call another programme independently of the platform (the *stdlib* function *system*). This approach, however, shows the usually unwanted behaviour to wait with any further execution until the called programme has finished.

The special function DefInput

DefInput

allows for inputting values for one or more EUROMOD variables from a text file.

The input file must be organised as a tab delimited table.

The function offers two modes,

which can be described as "look up mode" and "input mode".

The following examples are to illustrate what this means, starting with the "input mode".

Example 1: input mode

Policy	**SL_demo**	*Comment*
DefInput	**on**	
Path	c:\SomeFolder	
File	SomeInputFile.txt	
RowMergeVar	dgn	

File SomeInputFile.txt:

dgn	sin01_s	sin02_s	$SomeVar
0	4711	111	1234
1	1147	999	9876

Extract of Output:

idperson	dgn	sin01_s	sin02_s	$SomeVar
101	0	4711	111	1234
102	1	1147	999	9876
...	...	...	...	...
123401	0	4711	111	1234

This rather simple (and abstract)

example should be self-explaining. Note however, that the inputted variables must exist, i.e. be defined in the variables file (*sin01_s*, *sin02_s*) or by a EUROMOD function, e.g. *DefVar* (*$SomeVar*).

Moreover, if the input file contains a variable not known by EUROMOD an error message is issued. See the next example for ignoring columns (and rows) in the input file.

With respect to error messages,

note that all errors in context with *DefInput* (e.g. file does not exist, etc.) stop the programme run (i.e. are real errors and not just warnings). This is somewhat special as usually EUROMOD, where possible, avoids run time errors and tries to detect errors at read time. However, in the case of *DefInput*, for reasons of efficiency (see the

discussion of parameter *MultiSystemUse* below), inputting of variable values, as well as the reading of the input file, take place at run time in accordance to the position of the function in the spine.

Example 2: input mode

Policy	**SL_demo**	*Comment*
DefInput	**on**	
Path	c:\SomeFolder	
File	SomeInputFile.txt	
RowMergeVar	idperson	
DefaultIfNoMatch	999999	
IgnoreNRows	1	
IgnoreNCols	1	

File SomeInputFile.txt:

idperson	sin01_s	sin02_s	$SomeVar
101	123	321	132
123401	789	987	798

Extract of Output:

idperson	sin01_s	sin02_s	$SomeVar
101	123	321	132
102	999999	999999	999999
...	...	...	...
123401	789	987	798

This (again abstract) example

illustrates the use of the parameters *IgnoreNRows* and *IgnoreNCols*, which simply allow for ignoring headers in the input file. More importantly, the example demonstrates what happens if the input file does not contain a "match" for a specific person. As illustrated a default value can be indicated via the parameter *DefaultIfNoMatch*. If no such value is available, the programme issues an error message, if it cannot establish a match.

Also note that the function does not provide a *TAX_UNIT* parameter, meaning that it tries to establish a match for each row (person) of the EUROMOD input dataset. Nevertheless, it is possible to input variables on household level, as the following example shows.

Example 3: input mode

Policy	**SL_demo**	*Comment*
DefInput	**on**	
Path	c:\SomeFolder	
File	SomeInputFile.txt	
RowMergeVar	idhh	

File SomeInputFile.txt:

idperson	sin01_s	sin02_s	$SomeVar
101	123	321	132
123401	789	987	798
...	...	...	...

Extract of Output:

idhh	idperson	sin01_s	sin02_s	$SomeVar
1	101	123	321	132
1	102	123	321	132
1	103	123	321	132
2	201	789	978	798
...	...	...	...	...

The example demonstrates that one row of the input file can serve as input for several rows (persons) of the EUROMOD dataset. For obvious reasons the opposite is not possible, i.e. if the input file contains two rows with *idhh* set to 1, an error message is issued.

The next example demonstrates the "look up mode".

Example 4: look up mode

Policy	**SL_demo**	*Comment*
DefInput	**on**	
Path	c:\SomeFolder	

File	SomeInputFile.txt	
RowMergeVar	drg	
ColMergeVar	dgn	
InputVar	sin01_s	
IgnoreNRows	1	
IgnoreNCols	1	

File SomeInputFile.txt:

region	**gender**	
	0	**1**
1	10	11
2	20	21
3	30	31

Extract of Output:

idperson	**drg**	**dgn**	**sin01_s**
101	1	0	10
201	1	1	11
301	2	0	20
401	3	1	31
...	...	...	...

In the "look up mode"

only one EUROMOD variable is inputted, the parameter *InputVar* (*sin01_s* in the example) describes which. The (new) value of this variable is determined by the crossing point of a certain row and column. The respective row is found by searching the row in the input file whose header corresponds to the value of a person's variable defined by the parameter *RowMergeVar* (*drg*, i.e. region, in the example). Accordingly the respective column is found by searching the column in the input file whose header corresponds to the value of a person's variable defined by the parameter *ColMergeVar* (*dgn*, i.e. gender, in the example).

Note the parameters *IgnoreNRows* and *IgnoreNCols* and, what's more, that *drg* respectively *dgn* are not named anywhere in the input file, but solely indicated by the parameters *RowMergeVar* and *ColMergeVar*.

That means the programme expects the values of these variables in the first (not ignored) row and column.

The following example shows the use

of the parameter *DoRanges*.

Example 5: look up mode

Policy	**SL_demo**	*Comment*
DefInput	**on**	

Path	c:\SomeFolder	
File	SomeInputFile.txt	
RowMergeVar	dag	
ColMergeVar	dgn	
InputVar	sin01_s	
DoRanges	yes	

File SomeInputFile.txt:

age/gender	0	1
10	100	101
30	300	301
80	800	801
200	2000	2001

Extract of Output:

idperson	drg	dgn	sin01_s
101	31	1	801
102	30	0	300
103	2	0	100
201	99	1	2001
...	...	...	...

If the

parameter *DoRanges* is set to *yes*, the values of *RowMergeVar* and *ColMergeVar* (in the case of "look up mode") in the input file are interpreted as upper limit of a range. That means, in the example, for all women aged up to 10 (*dag*<=10) *sin01_s* is set to 100. For all women older than 10 and aged up to 30 (10<*dag*<=30) *sin01_s* is set to 300, etc.[1] Note that, without the last range of 200, an error message would be issued for all persons older than 80, as no range applies and *DefaultIfNoMatch* is not defined.

The next parameter

to discuss, *MultiSystemUse*,

has no impact on the behaviour of the function but only concerns efficiency. If it is set to *yes* (the default), the

input table is kept in memory until the programme terminates. If set to *no*, this memory is released as soon as the content is assigned to EUROMOD variables. In other words, the memory for the input table is allocated once the function has its turn in the spine and immediately released after. If there is only one use of one system of the input file, the latter approach is of course most efficient. If however several systems use the input file, reading the file for each of them is probably quite inefficient. Setting *MultiSystemUse* to no may however even make sense if more than one system uses *the* input data, i.e. when the content of the file changes between the different uses (the *the* is set to italics to indicate that it is in fact not the same data). Finally note that using the input file by several systems does not necessarily lead to the same result, as the systems' values of *RowMergeVar* and *ColMergeVar* may differ.

The last example is a more extended

use of the function and intends to demonstrate how the three functions *DefOuput*, *CallProgramme* and *DefInput* can

be used together to allow an external programme doing some work for EUROMOD.

Example 6:

Policy	**SL_demo**	*Comment*
DefOutput	**on**	
File	InputForSomeProgramme.txt	
var	idhh	
var	idperson	
il	ils_origy	
il	ils_dispy	
TAX_UNIT	tu_individual_sl	
CallProgramme	**on**	
Path	c:\SomeFolder	
Programme	SomeProgramme.exe	
Argument	&Output\InputForSomeProgramme.txt	
RepByEMPath	&	
Wait	yes	
DefInput	**on**	
Path	&Output	
File	OutputFromSomeProgramme.txt	
RepByEMPath	&	
RowMergeVar	idperson	

In the example, firstly (*DefOutput*)

EUROMOD produces some output, consisting of household and person id as well as original and disposable income, and stores it in the file *InputForSomeProgramme.txt* located in the EUROMOD output folder.

Secondly (*CallProgramme*),

a programme called *SomeProgramme.exe* located at *c:\SomeFolder* is called.

The programme is assumed to take its input file as first argument and requiring exactly the output just produced by EUROMOD. Note the parameter *RepByEMPath*, which allows for addressing the EUROMOD output folder. EUROMOD waits (parameter *Wait* set to yes) for the programme

terminating its work. Whatever that may be in general, anyhow the programme is assumed to store its output in a file called *OutputFromSomeProgramme.txt* in the EUROMOD output folder.

Finally (*DefInput*)

EUROMOD reads this file and does whatever appropriate with its content.

[1] In technical terms the range mode is also applied on the column variable gender, but in practical terms this does not matter (as the = in <=

applies).

The special function AddHHMembers

This function is only available with EM3.

As the name suggests, the function AddHHMemembers allows adding new members to households. To get an idea, let’s start with an example:

Example 1:

Policy	*Grp/No*	**SL_demo**	*Comment*
AddHHMembers		**on**	
Add_Who	1	Partner	
PartnerCond	1	{ dag = 30 } & { dgn = 1 } & { idpartner <= 0 }	
dag	1	30	
dgn	1	0	

In this example, each household containing a person fulfilling the condition, i.e. to be a 30 year old male without a partner, gets a new female member, which is 30 years old and will be the partner of the 30 year old male.

There are in fact three modes of adding household members:

Modes of adding

These are determined by the parameter 'Add_Who', which can take the values 'Partner', 'Child' or 'Other'.

Adding partners: As the example above illustrates, 'Add_Who = Partner' requires a 'PartnerCond'.

If this condition is fulfilled by any (existing) household member, a new person is added to the household.
The variable idPartner of the new person is (system-)set to the idPerson of the "triggering" household member (i.e. the household member fulfilling the condition).

However, whether this is also true vice-versa, i.e. whether the idPartner of the triggering person is set to the idPerson of the new person, depends.

If the triggering person already has a partner, it keeps this partner. Note that this means that (s)he has now a main partner (the old one) and a secondary partner (the new one).

The model may comment this by warnings: 'more than one possible partner found …'. The example above avoids this by the condition { idpartner <= 0 }.
Maybe this is obvious, but it may be worth mentioning, that more than one person can be added to the household upon this 'Add_Who'.

In fact the number of new persons corresponds to the number of (existing) persons in the household, who fulfil the condition.
How to further specify the new persons, except from couple issues, i.e. determine their age, gender, etc., is explained under paragraph 'Characterising new persons and system-set variables'.

Adding children is in many aspects similar to adding partners, thus we can concentrate on the differences.

First, one needs to set 'Add_Who = Child' (instead of 'Partner') and use 'ParentCond' (instead of 'PartnerCond').
This time, the triggering person becomes the parent of the new child. That means, if the triggering person is female, the new child's idMother is set to her idPerson.

Then again, if the triggering person is male, the new child's idFather is set to his idPerson.
It is tried (by default) to find a second parent for the child. That means, if the first parent (the triggering person) has a partner (defined by idPartner) and this partner does not have the same gender as the first parent, the child's other parent-id (i.e. either idMother or idFather) is set to this idPartner.

One can avoid this behaviour, i.e. generate "single parent children", by setting the parameter 'IsPartnerParent' to 'no'.
Again the number of children added by this 'Add_Who' corresponds to the number of (existing) persons in the household, who fulfil the condition.
It is also probably necessary to further specify the new children, otherwise they are zero year old girls, but this is explained under paragraph 'Characterising new persons and system-set variables'.

Adding other persons: 'Add_Who = Other' allows to add other persons to the household than partners and children.

In this case one needs to use the 'HHCond' (instead of the 'PartnerCond' or 'ParentCond').

As the name suggests this condition must be fulfilled by the household and not by single persons.

Thus there are no triggering persons and the 'Add_Who' adds one person if the condition is fulfilled and no person if not.
There is more about the concrete meaning of "the household must fulfil the condition" under paragraph 'Taxunits', and more about specifying the new person under paragraph 'Characterising new persons and system-set variables'.

"Characterising" new persons and system-set variables

There are seven variables which are system-set: idHH, idPerson, dwt, dct, idFather, idMother and idPartner.

The last three depend on the 'Add_Who' parameter as described above. Household-id (idHH), weight (dwt) and country (dct) are copied from existing household members (as there is no question about their value). The person-id (idPerson) is just the next available id, i.e. max(idPerson) in household plus one.

Any other variable is initially set to zero, but one can specify another value as illustrated in the examples above and below. One can use formulas for doing so, but it's probably good to know on which level they operate – this is explained in more detail under paragraph 'Taxunits'.

Maybe not very relevant, but just to mention, one cannot initialise variables that are not used anywhere else. For example, if one tries to set stm12_s = 4711, but stm12_s is not used anywhere else, this would lead to the error message '… variable stm12_s does not exist.'

Maybe a bit more relevant, but only for tricky cases: the programme does not allow to "manually" initialise idHH, idPerson, dwt and dct.

It is however possible to do this for idPartner, idMother and idFather.

'Add_Who' groups

In fact one can add partners, children and other persons within the same AddHHMembers function,

because the parameter 'AddWho' plus the appropriate condition parameter plus the parameters for variable specification form groups.

The following example, an extension of the initial example, may illustrate this:

Example 2:

Policy	*Grp/No*	**SL_demo**	*Comment*
AddHHMembers		**on**	
Add_Who	1	Partner	
PartnerCond	1	{ dag = 30 } & { dgn = 1 } & { idpartner <= 0 }	
dag	1	30	
Add_Who	2	Child	
ParentCond	2	{ dag = 30 } & { dgn = 1 } & { idpartner <= 0 }	
dag	2	6	

Now the 30 year old males do not only get 30 year old partners but in addition 6 year old daughters.

In fact this example may be good for illustration but less for real application, because one probably wants the new partners to be the mothers of the new children.

This would not be the case, because it is rather important to note that any condition and formula refers to the initial household.

In the initial household the males do not have partners, thus the children would only have a father.

This can be easily changed by just using two subsequent AddHHMembers functions, where in the first the partners are added and in in the second the children.

A more meaningful application of groups may be to add more than one child to a person.

Just a hint: it may be a good idea to take special care of correct group-number-setting.

The programme tries of course to detect errors, but some may not be evident for a machine.

Taxunits

In the explanations above units of assessment were rather imprecisely referred to as individual or household, and in principle that is what is true for the function AddHHMembers - it does not use specific taxunits, but two simple and not further specified taxunits, one for individuals and one for households. An example may illustrate the consequences.

Let’s assume we want to add an au-pair to each household that has three children or more by this:

Example 3:

Policy	*Grp/No*	**SL_demo**	*Comment*
AddHHMembers		**on**	
Add_Who	1	Other	
HHCond	1	{ nDepChildrenInTu >= 3 }	
dag	1	19	

The above has absolutely no effect, because no household has children as the simple default household has no child definition and as a consequence nobody is a child.

The following will work for the purpose described above, because the OECD household has a definition of children (age<14).

Example 4:

Policy	*Grp/No*	**SL_demo**	*Comment*
AddHHMembers		**on**	
Add_Who	1	Other	
HHCond	1	{ nDepChildrenInTu#2 >= 3 }	
#_Level	2	tu_hh_oecd_co	
dag	1	19	

The motivation to implement it like this was to not overload the function and outsource any complicated taxunit issues to other functions, which can overtake preparation work for the AddHHMembers function. Let's now fully specify the rules: **Rule 1:** The conditions 'PartnerCond' and 'ParentCond' use a simple individual taxunit.

On the one hand, this means that some queries may not work as intuitively assumed, on the other hand one can change to any other taxunit with the required specifications.

In the example below each family as defined by tu_sben_family_sl gets a new 0 year old girl (as default dag and dgn are set to 0), provided the "family" has a female partner aged between 30 and 35 and the family does not yet have more than one child.

Example 5:

Policy	*Grp/No*	**SL_demo**	*Comment*
AddHHMembers		**on**	
Add_Who	1	Child	
ParentCond	1	{IsPartner#2} & {dgn=0} & {dag>=30} & {dag<=35} & {nDepChildrenInTu#2<=1}	
#_Level	2	tu_sben_family_sl	

Rule 2: The condition 'HHCond' and the formulas for initialising variables use a simple household taxunit[1].

As the au-pair example above illustrates, this still allows changing to another household taxunit which provides the required specifications.

It is however not possible to change to smaller units (this is the rule for the '#_Level' parameter, which does not allow changing to smaller units than the default unit).

If necessary, one can use other functions and features to prepare for the AddHHMembers function.

Memo of points to take care of

This memo may help to remember the important and/or not immediately intuitive points.

- Parameters 'ParentCond' and 'PartnerCond' use a simple individual taxunit. One can change to any other taxunit using parameter '#_level'.

  Queries may require usage with care.
- Parameter 'HHCond' and formulas for initialising variables use a simple household taxunit. One can only change to other household taxunits using parameter '#_level'.

  More complicated taxunit operation may require preparation by using other functions.
- All conditions and formulas refer to the original household, before adding

any new person (where original means at the time before the function is carried out).

- Variables idHH, idPerson, dwt and dct are system-set.
- If parameter 'Add_who = Partner', the following holds:
  idPartner of new person = idPerson of person fulfilling 'PartnerCond'
  idPartner of fulfilling person = idPerson of new person, if idPartner <= 0
- If parameter 'Add_who = Child', the following holds:
  idMother of new child = idPerson of person fulfilling 'ParentCond' if she is female
  idFather of new child = idPerson of person fulfilling 'ParentCond' if he is male
  If parameter 'IsPartnerParent = yes' (default) the not yet set parent-id (idMother/idFather) is set to the idPartner of the person fulfilling 'ParentCond', provided idPartner > 0 and the gender of the partner is different from the first parent.
- Take care of setting group-numbers appropriately.

A final note, which is actually more or less only a technical detail, but maybe worth knowing.

Using the function AddHHMembers will usually prevent full parallel runs.

This is visible in the run-log, which does not show percentages but names of functions.

Apart from that the run may be slightly slower.

The only way to allow for full parallel run is to use the function either at the very beginning of the spine or immediately before the output policy.

[1] The unit just contains all HH-members, with the first being the head. There are no other specifications.

The special function Break

The function Break allows the user to break the run at any point inside the spine. This function

is only intendent to be used during the development stage of a system to help the developer locate and

fix bugs in such system. It is not intendent to be included in published models.

This function is only available with EM3.

Example 1:

Policy	*Grp/No*	**SL_demo**	*Comment*
Break		**on**	

In the above example the execution will immediately stop once the Break function is reached, and EUROMOD

will produce an output file (taking the name from the first DefOutput to be found after the Break function,

which will typically be the default output function) that contains ALL variables and incomelists available at the time.

Note that outputting incomelists will very likely lead to issuing warnings "... use of not initialised variable ...",

referring to variables which are included in the incomelists, but only generated after the break.

A more controlled behaviour is shown in the next example:

Example 2:

Policy	*Grp/No*	**SL_demo**	*Comment*
Break		**on**	
ProduceOutput		yes	
OutputFileName		C:\MyFiles\debugging.txt	
ProduceTUinfo		yes	

In example 2, the execution will immediately stop once the Break function is reached, as before, but EUROMOD will this time produce an output file in the specified location and name. If only the filename is specified,

then it will be generated in the default output folder (same as the DefOutput works). This file will not only

contain variables and incomelists, but also all at the time available TU info.

EUROMOD add-ons and the special functions AddOn_Applic, AddOn_Pol, AddOn_Func and AddOnPar

Add-on features are EUROMOD extended functionalities, which

are not part of the standard tax-benefit calculations. The main reasons for keeping add-ons separate from the basic model, is to hold the latter as clear and straight forward as possible. Typical examples for add-ons are the calculation of marginal/average tax-rates; the calculation of disposable income by different labour supply options; templates for optimisation exercises, e.g.

choices between different policy options; templates for budget neutral reforms, etc. While these examples are usually implemented for all or at least for a couple of countries, add-on features can also be used to store reform scenarios separately.

Technically add-on implementations look like country

implementations, i.e. they consist of systems, policies, functions, etc. There are some minor differences and particularities:

- Add-ons are loaded in the user interface via the

  ribbon Add-Ons.
- The functions AddOn_Applic, AddOn_Pol, AddOn_Func and AddOnPar can only be used with add-ons, where the existence of AddOn_Applic is compulsory (see below).
- Add-ons make extensive use of

  "identifiers"[1]: see examples below and EUROMOD Functions – Identifiers and the placeholders =cc= and =sys=.
- Add-ons make extensive use of the placeholders

  =cc= and =sys=[1]: see examples below and EUROMOD Functions – Identifiers and the placeholders =cc= and =sys=.
- Add-on parameters are freely editable. That

  means for example that, other than for countries, on the one hand any text can be indicated for taxunit parameters, on the other hand there is no support in form of offering available taxunits.

This is based on the fact that add-ons use components (i.e., e.g. taxunits) which are not defined in the add-on itself but can be defined in any country. Thus the usual control and support would be technically too complex.

Example

Example 1 aims to illustrate the concept of add-ons, by

showing a simple application.

Example 1: policy Definition_ex1

Policy	**Ex1**	**Ex1_MT**
AddOn_Applic	**on**	**on**
Description	Add-ons example 1	Add-ons example 1 for Malta
Sys	*2010*	MT_2010
Sys	sl_demo	n/a
SysNA	MT*	n/a
AddOn_Pol	**on**	**on**
Pol_Name	bchot_ex1	bchot_ex1
Insert_Before_Pol	output_std_=cc=	output_std_mt
AddOn_Func	**on**	**on**
Id_Func	2b3e7a88-0081-4500-bc13-e902923af3e1	5674e105-6f33-433d-8c8b-db591b53c4f9
Insert_After_Func	output_std_=cc=_#1	2AB9FACE-08AD-492F-BC49-47CF3ADAB6EC
AddOn_Par	**on**	**on**
Insert_Func	output_std_=cc=_#1	2AB9FACE-08AD-492F-BC49-47CF3ADAB6EC
Var	bchot_s	bchot_s
IL	ils_dispy_ext	ils_dispy_ext

Example 1: policy Implementation_ex1

Policy	**Ex1**	**Ex1_MT**
DefOutput	**on**	**on**
File	bchot_=cc=.txt	bchot_MT.txt
Var	idhh	idhh
VarGroup	bchot*	bchot*
ILGroup	il_bchot*	il_bchot*
TAX_UNIT	tu_bchot	tu_bchot

Example 1: policy bchot_ex1

Policy	**Ex1**	**Ex1_MT**
DefVar	**on**	**on**
bchot_nCh_upTo6	0	0
bchot_nCh_6plus	0	0
bchot_DayCare_Fee	0#d	0#d

...	...	...
DefTU	**on**	**on**
Name	tu_bchot	tu_bchot
Type	SUBGROUP	SUBGROUP
...	...	...
DefIL	**on**	**on**
Name	ils_dispy_ext	ils_dispy_ext
ils_dispy	+	+
bchot_s	+	+
DefIL	**on**	**on**
Name	il_bchot_means	il_bchot_means
...	...	...
...	**...**	**...**
...	...	...
ArithOp	**on**	**on**
who_must_be_elig	...	...
formula	...	...
output_var	bchot_s	bchot_s
TAX_UNIT	tu_bchot	tu_bchot

The example contains the

implementation of an "other child benefit" (*bchot_s*), which can be added to any 2010 tax-benefit systems implemented in EUROMOD, as well as to Simpleland. It uses three policies for this purpose. The first, *Definition_ex1*, can be

regarded as the centrepiece as it controls what happens. The second, *Implementation_ex1*, contains add-on specific implementations and the third, *bchot_ex1*,

implements the "other child benefit". The add-on has two systems. The first, *Ex1*, can be applied on all

countries except Malta. The second, *Ex1_MT*,

is especially designed for the Maltese 2010 system and illustrates what's different if one knows the country and system on which the add-on is applied. In the following the "systems on which the add on is

applied" will be referred to as base systems for brevities sake.

The policy *Definition_ex1* contains four functions. The first, *AddOn_Applic*, specifies on which base systems the add-on systems can be applied. The first *Sys* parameter allows for all 2010

systems (more precise, system names containing "2010"), the second *Sys* parameter allows for Simpleland's *sl_demo* system. The SysNa parameter (NA stands for not applicable) excludes any Maltese systems (more precise, system names starting with "MT").

The second function, *AddOn_Pol,*

defines that the policy *bchot_ex1* is

to be added to the base system, i.e. the policy implementing the "other child benefit". This is accomplished by the parameter *Pol_Name*, while the parameter *Insert_Before_Pol* specifies where to integrate the policy. The child benefit is inserted before the standard output policy, which in the Maltese system, *Ex1_MT*, can be simply indicated as *output_std_mt*, while the many country system, *Ex1*, needs to use the

placeholder =cc=, which is replaced by the respective country's acronym ant run-time (see EUROMOD Functions – Identifiers and the placeholders =cc= and =sys=).

The third function, *AddOn_Func*,

defines that a special output function is to be added to the base system, i.e.

the one defined in the policy *Implementation_ex1*.

The parameter *Id_Func* accomplishes this task by indicating the identifier of this function (see EUROMOD Functions – Identifiers and the placeholders =cc= and =sys=). Note that, though seemingly identifying the same function, the identifier is not equal for the two systems *Ex1* and *Ex1_MT*. The reason is that identifiers are system specific and in the strict sense the two identifiers do not refer to the same function, as its implementation may be different for the two systems. The parameter *Insert_After_Func* specifies where to integrate the function, which is for both systems "after the function defining standard output". However, the system *Ex1* uses a symbolic identifier for this purpose (see EUROMOD Functions – Identifiers and the placeholders =cc= and =sys=), which is composed of the name of the policy where the function is located, i.e. the standard output policy (*output_std_=cc=*), and the order of the function, i.e. the function is supposed to be the first (*#1*) in this policy. This is in fact not a unique identifier and therefore somewhat uncertain, which must however be accepted if the definition is to be valid for several base systems. Such uncertainty can be avoided if the add-on system refers to a single base system,

therefore *Ex1_MT* uses the unique

identifier of the standard output function in the Maltese 2010 system.

The fourth function, *AddOn_Par,* adds

the two main outputs of the "other child benefit" implementation to standard output, i.e. the benefit itself (*Var* = *bchot_s*) and an incomelist defining standard disposable income extended by the benefit (*IL* = *ils_dispy_ext*).

Note that the parameter *Insert_Func* is set to the same values as the parameter *Insert_After_Func* of the function *AddOn_Func*.

Less obviously note that it is important to insert the special output function after the standard output function. Using the parameter *Insert_Before_Func* instead of the parameter *Insert_After_Func* on first view would not do any harm, as it is not important which output comes first. However the insertion of the special output function would move the standard output function from first order to second order, thus *#1* would not work anymore. Obviously this only matters for the system *Ex1*,

which uses a symbolic identifier, while the "real" identifier of *Ex1_MT* would still work.

The policy *Implementation_ex1* contains only one function, namely the special output function referred to by the function *AddOn_Func*, outputting some intermediate variables and incomelists produced by

the "other child benefit" policy. Note, that this policy is in fact dispensable, as the function could as well be defined in the policy *Definition_ex1*. In fact the policy just serves illustration purposes as with more complex add-ons it might be more transparent to separate definition from implementation. More important, however note that there needs to be at least one policy which is not integrated into the base system. This policy needs to contain the *AddOn_X* functions and (usually) all functions integrated via *AddOn_Func*, as, if the latter were defined in an integrated policy, these functions would run twice (which may occasionally make sense).

Finally the policy *bchot_ex1* contains the implementation of the "other child benefit". It is partly left to the readers’ imagination what it actually does, denoted by the ... parts.

Generation and storage of add-ons

Physically add-ons are stored in xml files, which are named

like the add-on’s short name. SeeWorking with EUROMOD - Changing countries' names and short names for more information on countries' and add-ons' short names and EUROMOD Installation and Architecture for matters of storage.

Add-ons can be generated by using the *Save As* functionality (see Working with EUROMOD - Saving, saving as and auto-saving) to

copy a suitable existing add-on and adapt it accordingly. Possibilities to generate add-ons from scratch or derive them from differences between a base system and a (potential) add-on system implemented in some country are planned but not yet implemented.

Application of add-ons

For information on how to apply an add-on, i.e. merge it

with a base system and run the result, see the paragraph *Running add-on systems* in Working with EUROMOD - Running EUROMOD.

[1]Though identifiers and the placeholders =cc= and =sys=

are not add-on specific. The former are also used by other functions (e.g. the loop functions and *ChangeParam*).

The latter can be used universally in principle

Summary of functions and their parameters

This section provides a full (but brief) description of the parameters of EUROMOD functions. For a descriptive explanation with many examples see EUROMOD Functions - Description of functions and their parameters.

Summary of parameters for function ArithOp

A simple calculator, allowing for the most common arithmetical operations.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Formula	formula	no	yes	n/a	Formula for calculating the function's result. Allowed operations: - addition: operator + - subtraction: operator - - multiplication: operator * - division: operator / - raising to a power: operator ^, e.g. 2 ^ 3 (result: 8) - percentage: operator %, e.g. yem*3% (result: yem*(3/100)) - reminder of division: operator \, e.g. 22\5 (result: 2) - minimum and maximum: operators <min> and <max>, e.g. 10<min>15 (result: 10) - absolute value: operator <abs>(), e.g. <abs>(50-70) (result: 20) - negation: operator !(), e.g. !(IsMarried), !(17) (result: 0), !(0) (result: 1) Allowed operands: - numeric values, e.g. 10, 0.3, -25 - numeric values with a period, e.g. 12000#m, 1000#y - amount#i as place holders for numeric values specified by footnote parameters - variables, e.g. yem - incomelists, e.g. ils_dispy - queries, e.g. IsUnemployed - random numbers: rand, e.g. rand * 100 (result: random number between 0 and 100, see function RandSeed for more information) Order of operation rules: - ^, <min>, <max>, <abs>(), !(), % - before multiplicative operations */ \ - before additive operations +- Parentheses can be used to group operations, e.g. (2+3)*4.

Common Parameters

Name	Type	Optional	Single	Default	Description
Who_Must_Be_Elig	categorical	yes	yes	nobody	Function's calculations are carried out if ... - one (one_member): ... one member of the assessment unit is "eligible" - one_adult: ... one adult member of the assessment unit is "eligible" - all (all_members; taxunit): ... all members of the assessment unit are "eligible" - all_adults: ... all adult members of the assessment unit are eligible - nobody: ... always "eligible" is determined by the variable indicated by

					parameter Elig_Var
Output_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Output_Add_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function is added to the current value of the variable.
Result_Var	variable	yes	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Elig_Var	variable	yes	yes	sel_s	Variable indicating whether a person is "eligible" (see parameter Who_Must_Be_Elig): - zero: person is not eligible - non zero: person is eligible
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
LowLim	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller.
UpLim	formula	yes	yes	999999999999.99	Replaces result of function if result is higher.
Threshold	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller: if lower limit is not defined by zero, otherwise by lower limit.
#_LimPriority	categorical	yes	yes	upper	Footnote parameter for the further specification of an operand: Possible values: If upper limit (#_UpLim) is smaller than lower limit (#_LowLim) ... - upper: ... upper limit dominates; - lower: ... lower limit dominates; - not defined: ... a warning is issued.
Round_to	amount	yes	yes	n/a	Result is rounded to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Up	amount	yes	yes	n/a	Result is rounded up to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Down	amount	yes	yes	n/a	Result is rounded down to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.

#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.

Summary of parameters for function Elig

Is most frequently
used for determining the eligibility for receiving benefits. However, it also allows for determining the liability for paying taxes, as well as evaluating other conditions.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Elig_Cond	condition	no	yes	n/a	A combination of conditions: output variable is set to 1 if they are fulfilled, else to 0. The comibation consists of: - conditions which must be fulfilled, e.g. {dag<19}, - conditions which must not be fulfilled, e.g. !{IsMarried}, - combined by the logical operators & (and) and | (or), e.g. {dag<19} & !{IsMarried}, - (possibly) grouped by parentheses, e.g. {dag<15} | ({dag<19} & !{IsMarried}). A single condition has - either 1 component, usually a yes-no-query, e.g. {IsUnemployed} - or 2 components separated by a comparison operator >,<,>=,<=,=,!=, e.g. {poa=0}. The operands left and right from the comparison operator are: - numeric values, e.g. 10, 0.3, (-25), note that negative values must be bracketed - numeric values with a period, e.g. 12000#m, 1000#y - amount#i as place holders for numeric values specified by footnote parameters - variables, e.g. yem - incomelists, e.g. ils_dispy - queries, e.g. IsUnemployed

Common Parameters

Name	Type	Optional	Single	Default	Description
Output_Var	variable	yes	yes	sel_s	Variable for storing the result of the function. Result of function overwrites the current value of the variable.

Result_Var	variable	yes	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable. Note that Result_Var provides an **additional** variable to store the result. That means, even if it is used and the parameter *Output_Var* is not indicated, *sel_s* will be set to the result of the function.
Who_Must_Be_Elig	categorical	yes	yes	nobody	Function's calculations are carried out if ... - one (one_member): ... one member of the assessment unit is "eligible" - one_adult: ... one adult member of the assessment unit is "eligible" - all (all_members; taxunit): ... all members of the assessment unit are "eligible" - all_adults: ... all adult members of the assessment unit are eligible - nobody: ... always "eligible" is determined by the variable indicated by parameter Elig_Var
Elig_Var	variable	yes	yes	sel_s	Variable indicating whether a person is "eligible" (see parameter Who_Must_Be_Elig): - zero: person is not eligible - non zero: person is eligible
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.

#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function BenCalc

Allows for modelling a wide range of policy instruments, in particular benefits.
The result is calculated as a sum of "components", where the value of a component is only added if a certain condition is fulfilled by at least one member of the assessment unit.
The following stylised formulas illustrates the approach:
result = Sum (Comp_perTU if Comp_Cond = true)
result = Sum (Comp_perElig * nElig if Comp_Cond = true)

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Base	formula	yes	yes	n/a	Base amount that can be used with parameters compX_perTU / compX_perElig, referenced as $Base.
Comp_perTU	formula	no	yes	n/a	Formula to calculate one component of the function's result. The result of the formula is added once to the function's result (regardless whether one or more members of the assessment unit fulfil the components condition (Comp_Cond)). Syntax rules as for parameter Formula of function ArithOp apply.
Comp_perElig	formula	no	yes	n/a	Formula to calculate one component of the function's result. The result of the formula is added to the function's result once for each member of the assessment unit fulfiling the components condition (Comp_Cond). Syntax rules as for parameter Formula of function ArithOp apply.
Comp_Cond	condition	no	yes	n/a	Condition that must be fulfilled to add the component (comp_perTU / comp_perElig) to the function's result. Syntax rules as for parameter Elig_Cond of function Elig apply.
Withdraw_Base	formula	yes	yes	n/a	Withdraw_Base * Withdraw_Rate is deducted from function's (preliminary) result. Note that if withdraw parameters are used, the function's result cannot be negative.
Withdraw_Rate	formula	yes	yes	0	Withdraw_Base * Withdraw_Rate is deducted from function's (preliminary) result. Note that if withdraw parameters are used, the function's result cannot be negative.
Withdraw_Start	formula	yes	yes	0	Level of Withdraw_Base where withdrawal starts.
Withdraw_End	formula	yes	yes	999999999999.99	Level of Withdraw_Base where withdrawal ends (i.e. benefit is totally withdrawn). Note that the parameter is ignored if Withdraw_Rate is indicated.
Comp_LowLim	formula	yes	yes	-999999999999.99	Replaces component if component is smaller.
Comp_UpLim	formula	yes	yes	999999999999.99	Replaces component if component is higher.

Common Parameters

Name	Type	Optional	Single	Default	Description
Output_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Output_Add_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function is added to the current value of the variable.
Result_Var	variable	yes	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Who_Must_Be_Elig	categorical	yes	yes	nobody	Function's calculations are carried out if ... - one (one_member): ... one member of the assessment unit is "eligible" - one_adult: ... one adult member of the assessment unit is "eligible" - all (all_members; taxunit): ... all members of the assessment unit are "eligible" - all_adults: ... all adult members of the assessment unit are eligible - nobody: ... always "eligible" is determined by the variable indicated by parameter Elig_Var
Elig_Var	variable	yes	yes	sel_s	Variable indicating whether a person is "eligible" (see parameter Who_Must_Be_Elig): - zero: person is not eligible - non zero: person is eligible
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
LowLim	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller.
UpLim	formula	yes	yes	999999999999.99	Replaces result of function if result is higher.
Threshold	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller: if lower limit is not defined by zero, otherwise by lower limit.
#_LimPriority	categorical	yes	yes	upper	Footnote parameter for the further specification of an operand: Possible values: If upper limit (#_UpLim) is smaller than lower limit (#_LowLim) ... - upper: ... upper limit dominates; - lower: ... lower limit dominates; - not defined: ... a warning is issued.
Round_to	amount	yes	yes	n/a	Result is rounded to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Up	amount	yes	yes	n/a	Result is rounded up to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Down	amount	yes	yes	n/a	Result is rounded down to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.

#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function SchedCalc

Allows for the implementation of the most common (tax) schedules.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Base	formula	no	yes	n/a	Base income for the schedule.
Band_UpLim	formula	no	yes	n/a	Upper limit of band.
Band_LowLim	formula	no	yes	n/a	Lower limit of band.
Band_Rate	formula	no	yes	n/a	Rate to apply on band.
Band_Amount	formula	no	yes	n/a	Amount to add for band.
Do_Average_Rates	yes/no	yes	yes	no	If set to yes the rate of the highest band reached by Base is applied on the total amout of Base.
Quotient	formula	yes	yes	n/a	If defined Base is divided by the quotient before the schedule is applied. Afterwards the result is multiplied by the quotient.
BaseThreshold	formula	yes	yes	n/a	If Base is smaller result is set to zero.
Round_Base	amount	yes	yes	n/a	If defined Base is rounded to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Simple_Prog	yes/no	yes	yes	n/a	If set to yes the same rate/amount is applied on all income. The respective rate/amount is the one of the highest band the income falls into.

Common Parameters

Name	Type	Optional	Single	Default	Description
Output_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Output_Add_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function is added to the current value of the variable.
Result_Var	variable	yes	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Who_Must_Be_Elig	categorical	yes	yes	nobody	Function's calculations are carried out if ... - one (one_member): ... one member of the assessment unit is "eligible" - one_adult: ... one adult member of the assessment unit is "eligible" - all (all_members; taxunit): ... all members of the assessment unit are "eligible" - all_adults: ... all adult members of the assessment unit are eligible - nobody: ... always "eligible" is determined by the variable indicated by parameter Elig_Var
					Variable indicating whether a person is "eligible"

Elig_Var	variable	yes	yes	sel_s	(see parameter Who_Must_Be_Elig): - zero: person is not eligible - non zero: person is eligible
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
LowLim	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller.
UpLim	formula	yes	yes	999999999999.99	Replaces result of function if result is higher.
Threshold	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller: if lower limit is not defined by zero, otherwise by lower limit.
#_LimPriority	categorical	yes	yes	upper	Footnote parameter for the further specification of an operand: Possible values: If upper limit (#_UpLim) is smaller than lower limit (#_LowLim) ... - upper: ... upper limit dominates; - lower: ... lower limit dominates; - not defined: ... a warning is issued.
Round_to	amount	yes	yes	n/a	Result is rounded to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Up	amount	yes	yes	n/a	Result is rounded up to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Down	amount	yes	yes	n/a	Result is rounded down to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function Allocate

Allows for (re)allocating amounts (incomes, benefits, taxes) between members of assessment units.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Share	formula	no	yes	n/a	Amount to be (re)allocated between assessment unit members.
Share_Between	condition	yes	yes	n/a	Condition that must be fulfilled by a member of the assessment unit to be among the persons between who the amount is (re)allocated. Syntax rules as for parameter Elig_Cond of function Elig apply.
Share_All_IfNoElig	yes/no	yes	yes	yes	If no member of the assessment unit fulfils the condition defined by Share_Between: - if set to yes: amount is equally (re)allocated among members of the assessment unit. - if set to no: Output_Var is set to zero (respectively zero is added to Output_Add_Var).
Share_Prop	variable/incomelist	yes	yes	n/a	If the parameter is defined, (re)allocation is carried out in proportion to this variable/incomelist.
Share_Equ_IfZero	yes/no	yes	yes	no	If the variable/incomelist defined by Share_Prop is zero for all members of the assessment unit (who fulfil Share_Between): - if set to no: an error message is issued. - if set to yes: amount is equally (re)allocated among all members of the assessment unit (who fulfil Share_Between).
Ignore_Neg_Prop	yes/no	yes	yes	no	If the variable/incomelist defined by Share_Prop is negative it is ignored (i.e. it is considered to be zero).

Common Parameters

Name	Type	Optional	Single	Default	Description
Output_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Output_Add_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function is added to the current value of the variable.
Result_Var	variable	yes	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an

					operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function Min

A simple minimum calculator.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Val	formula	no	no	n/a	Values for which the minimum should be calculated.
Positive_Only	yes/no	yes	yes	no	If set to yes, negative and zero-values are ignored. If there is no positive value the result is 0.

Common Parameters

Name	Type	Optional	Single	Default	Description
Output_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Output_Add_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function is added to the current value of the variable.
Result_Var	variable	yes	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Who_Must_Be_Elig	categorical	yes	yes	nobody	Function's calculations are carried out if ... - one (one_member): ... one member of the assessment unit is "eligible" - one_adult: ... one adult member of the assessment unit is "eligible" - all (all_members; taxunit): ... all members of the assessment unit are "eligible" - all_adults: ... all adult members of the assessment unit are eligible - nobody: ... always "eligible" is determined by the variable indicated by parameter Elig_Var
Elig_Var	variable	yes	yes	sel_s	Variable indicating whether a person is "eligible" (see parameter Who_Must_Be_Elig): - zero: person is not eligible - non zero: person is eligible
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
LowLim	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller.
UpLim	formula	yes	yes	999999999999.99	Replaces result of function if result is higher.
Threshold	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller: if lower limit is not defined by zero, otherwise by

					lower limit.
#_LimPriority	categorical	yes	yes	upper	Footnote parameter for the further specification of an operand: Possible values: If upper limit (#_UpLim) is smaller than lower limit (#_LowLim) ... - upper: ... upper limit dominates; - lower: ... lower limit dominates; - not defined: ... a warning is issued.
Round_to	amount	yes	yes	n/a	Result is rounded to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Up	amount	yes	yes	n/a	Result is rounded up to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Down	amount	yes	yes	n/a	Result is rounded down to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function Max

A simple maximum calculator.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Val	formula	no	no	n/a	Values for which the maximum should be calculated.
Positive_Only	yes/no	yes	yes	no	Makes only sense for function Min.

Common Parameters

Name	Type	Optional	Single	Default	Description
Output_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Output_Add_Var	variable	no	yes	n/a	Variable for storing the result of the function. Result of function is added to the current value of the variable.
Result_Var	variable	yes	yes	n/a	Variable for storing the result of the function. Result of function overwrites the current value of the variable.
Who_Must_Be_Elig	categorical	yes	yes	nobody	Function's calculations are carried out if ... - one (one_member): ... one member of the assessment unit is "eligible" - one_adult: ... one adult member of the assessment unit is "eligible" - all (all_members; taxunit): ... all members of the assessment unit are "eligible" - all_adults: ... all adult members of the assessment unit are eligible - nobody: ... always "eligible" is determined by the variable indicated by parameter Elig_Var
Elig_Var	variable	yes	yes	sel_s	Variable indicating whether a person is "eligible" (see parameter Who_Must_Be_Elig): - zero: person is not eligible - non zero: person is eligible
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
LowLim	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller.
UpLim	formula	yes	yes	999999999999.99	Replaces result of function if result is higher.
Threshold	formula	yes	yes	-999999999999.99	Replaces result of function if result is smaller: if lower limit is not defined by zero, otherwise by lower limit.

#_LimPriority	categorical	yes	yes	upper	Footnote parameter for the further specification of an operand: Possible values: If upper limit (#_UpLim) is smaller than lower limit (#_LowLim) ... - upper: ... upper limit dominates; - lower: ... lower limit dominates; - not defined: ... a warning is issued.
Round_to	amount	yes	yes	n/a	Result is rounded to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Up	amount	yes	yes	n/a	Result is rounded up to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
Round_Down	amount	yes	yes	n/a	Result is rounded down to nearest whole number if set to 1, to nearest number with 1 decimal if set to 0.1, to nearest 10 if set to 10, etc.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function Uprate

Allows for the uprating of monetary dataset variables.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Dataset	string	no	no	n/a	Name of a dataset(s) for which the uprating settings apply. If the settings apply for several datasets, the parameter can be used more than once. Moreover, the wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. be_20*_v?).
Def_Factor	string	yes	yes	n/a	Factor, which is applied on all monetary variables in the dataset, which do not have a particular own factor. Possible values are amounts (e.g. 1.023) or factors as defined by parameters Factor_Name, Factor_Value, Factor_Condition.
Factor_Name	string	yes	yes	n/a	Name of a specific uprating factor, whose value is specified by parameter Factor_Value.
Factor_Value	string	yes	yes	n/a	Value of a specific uprating factor. If no name is specified via Factor_Name, the factor can be addressed as factorX, where X refers to the number in the Grp/No column.
[Placeholder]	string	yes	no	n/a	[Placeholder] stands for the name of a variable, defined in the policy column, whose uprating factors are specified in the respective system columns. The uprating factors can be indicated as amounts (e.g. 1.023) or factors, as defined by parameters Factor_Name, Factor_Value, Factor_Condition.
Factor_Condition	condition	yes	yes	n/a	Condition that needs to be fulfilled to apply the factor specified by parameters Factor_Value and (optionally) Factor_Name.
AggVar_Name	variable	yes	yes	n/a	Name of an aggregate variable (e.g. yse).
AggVar_Part	variable	yes	no	n/a	Name of a component variable of the aggregate variable defined by AggVar_Name (e.g. ysebs).
AggVar_Tolerance	amount	yes	yes	0	If the value of the variable specified by parameter AggVar_Name differs from the sum of its components defined by parameters AggVar_Part, an error message is issued, but only if the absolute difference is higher than aggvarX_tolerance.
WarnIfNonMonetary	yes/no	yes	yes	yes	If set to yes, a warning is issues for uprating non-monetary variables.
WarnIfNoFactor	yes/no	yes	yes	yes	If set to yes an warning is issues for any monetary dataset variable without an explicitly defined uprating factor.
DBYearVar	string	yes	yes		If your dataset contains multiple years, use this parameter to specify which variable holds the DB year in the input data.
RegExp_Def	string	yes	yes	n/a	Pattern (regular expression) defining the group of variables to be uprated by RegExp_Factor, e.g. x[0-9]+.
RegExp_Factor	string	yes	yes	n/a	Factor by which the variables defined by RegExp_Def are uprated.

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.

Summary of parameters for function SetDefault

Allows for the setting of default values for not existent dataset variables.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Dataset	string	no	no	n/a	Name of a dataset(s) for which the default settings apply. If the settings apply for several datasets, the parameter can be used more than once. Moreover, the wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. be_20*_v?).
[Placeholder]	formula	yes	no	n/a	[Placeholder] stands for the name of a variable, defined in the policy column, whose default values are specified in the respective system columns (i.e. the default may be system specific).

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.

Summary of parameters for function DefIl

Allows for the definition of incomelists.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Name	string	no	yes	n/a	Name of the incomelist.
[Placeholder]	string	yes	no	n/a	[Placeholder] stands for the name of a component (variable or incomelist) of the incomelist, which is defined in the policy column. A plus (+) in the respective system's column means that the component is added. A minus (-) indicates that the component is subtracted. Also, a factor can be used, e.g. +3 means that the component is added 3 times.
Warn_If_NonMonetary	yes/no	yes	yes	yes	If yes, a warning is issued if any component is non-monetary.
RegExp_Def	string	yes	no	n/a	Pattern (regular expression) defining a group of variables, which should become components of the incomelist, e.g. x[0-9]+.
RegExp_Factor	string	yes	no	+	A plus (+) means that the variables matching the respective RegExp_Def are added to the incomelist. A minus (-) indicates that the variables are subtracted. Also, a factor can be used, e.g. +3 means that the variables are added 3 times. If the parameter is omitted, variables are added.

Common Parameters

Name	Type	Optional	Single	Default	Description

Summary of parameters for function DefTu

Allows for the definition of assessment units.
Note that parameters may use variables with the prefixes "head:" or "partner:". These prefixes can be used with variables only, not with incomelists or queries.
Also note that "{Default}" can be used to further define any default condition (as indicated in Default Value).

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Type	categorical	no	yes	n/a	Indicates the scope of the assessment unit. Possible values: - HH: all members of the household belong to one unit. - IND: each members of the household forms an own unit. - SUBGROUP: unit members are specified as indicated in the parameter Members.
Name	string	no	yes	n/a	Name of the assessment unit.
Members	string	yes	yes	n/a	Specifies which members of the household form a unit, if parameter Type is set to SUBGROUP. Syntax: Status & Status & Status & ..., e.g. Partner & OwnChild. Possible values for Status: - Partner: defined by parameter PartnerCond - OwnDepChild: defined by parameter OwnDepChildCond - LooseDepChild: defined by parameter LooseDepChildCond - LooseDepChild: defined by parameter LooseDepChildCond - OwnChild: defined by parameter OwnChildCond - DepParent: defined by parameter DepParentCond - DepRelative: defined by parameter DepRelativeCond Note, that the Head is obviously always part of unit and (usually) relations are defined with reference to the Head.
HeadDefInc	variable or incomelist	yes	yes	ils_OrigY	Incomelist used for determining who is the richest person in the assessment unit, see parameter ExtHeadCond.
					Condition further defining the head of the assessment unit. The condition is &-

ExtHeadCond	condition	yes	yes	!{IsDepChild}	linked with the following fixed head condition: {HeadDefInc>anyother:HeadDefInc} | ({HeadDefInc>=anyother:HeadDefInc} & {dag>anyother:dag}) | ({HeadDefInc>=anyother:HeadDefInc} & {dag>=anyother:dag} & {idperson<anyother:idperson}) where 'anyother' refers to potential heads, in the sense of fulfilling ExtHeadCond.
PartnerCond	condition	yes	yes	{head:idperson=idpartner}	Condition defining who is a partner.
DepChildCond	condition	yes	yes	{0}	Condition defining who is a dependent child. The "real" default, i.e. if the parameter is not defined or set to n/a, is {0} (i.e. nobody is a child). However, setting the parameter to "{Default}" is interpreted as !{isparent}&{idpartner<=0}.
OwnChildCond	condition	yes	yes	{head:idperson=idmother}| {head:idperson=idfather}| {partner:idperson=idmother}| {partner:idperson=idfather}	Condition defining who is an own child.
OwnDepChildCond	condition	yes	yes	{isownchild}&{isdepchild}	Condition defining who is an own dependent child.
LooseDepChildCond	condition	yes	yes	{idmother=0}&{idfather=0}&{isdepchild}	Condition defining who is a loose dependent child.
DepParentCond	condition	yes	yes	{head:idmother=idperson}| {head:idfather=idperson}| {partner:idmother=idperson}| {partner:idfather=idperson}	Condition defining who is a dependent parent.
DepRelativeCond	condition	yes	yes	{0}	Condition defining who is a dependent relative.
LoneParentCond	condition	yes	yes	{isparentofdepchild}&{idpartner<=0}	Condition defining who is a lone parent.
StopIfNoHeadFound	yes/no	yes	yes	no	If set to yes: an error is issued if ExtHeadCond rules out all household members. If se to no: no error issued, instead ExtHeadCond is dropped for affected households.
NoChildIfHead	yes/no	yes	yes	no	If set to yes (possible) child status is removed if person is the Head of the assessment unit.
NoChildIfPartner	yes/no	yes	yes	no	If set to yes (possible) child status is removed if person is Partner as defined by parameter PartnerCond.
AssignDepChOfDependents	yes/no	yes	yes	no	If set to yes dependent children of dependent unit members (i.e. persons who are not Head or Partner of the unit) are assigned to the unit.
AssignPartnerOfDependents	yes/no	yes	yes	no	If set to yes partners of dependent unit members (i.e. persons who are not Head or Partner of the unit) are assigned to the unit.
					Allows customizing the behaviour when multiple partners are found in the data.

MultiplePartners	categorical	yes	yes	warn	Possible values: - warn: Gives out a warning when multiple partners are found, and keeps only the first one. - ignore: Keeps only the first partner found, without giving out any warnings. - allow: Keeps all partners. The head still connects only to the first partner found, but all partners have IsPartner set to true.
IsStatic	yes/no	yes	yes	yes	If set to no, TU is recreated on each use. Set to 'no' e.g. for a child-definition depending on a variable that changes during run.

Common Parameters

Name	Type	Optional	Single	Default	Description
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function UpdateTu

Allows for the re-assessment of assessment units.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Name	taxunit	no	yes	n/a	Name of the assessment unit.
Update_All	yes/no	no	yes	n/a	If set to yes all assessment units are updated.

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.

Summary of parameters for function DefOutput

Allows for the definition of model output.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
File	string	no	yes	n/a	Name of text file to write the output to (the extension .txt can be omitted).
Var	variable	yes	no	n/a	Name of a variable to be included in the output.
VarGroup	string	yes	no	n/a	Regular expression that describes a group of variables to be included in the output, where * stands for any character and ? stands for one arbitrary character (e.g. b* includes all variables starting with b).
IL	incomelist	yes	no	n/a	Name of an incomelist to be included in the output. The total value of the incomelist is outputted (see parameter DefIL for outputting components).
ILGroup	string	yes	no	n/a	Regular expression that describes a group of incomelists to be included in the output, where * stands for any character and ? stands for one arbitrary character (e.g. ils* for all system incomelists).
DefIL	incomelist	yes	no	n/a	Name of an incomelist to be included in the output. The content of the incomelist is outputted, i.e. the variables contained in the incomelist (see parameter IL for outputting the total value).
UnitInfo_Id	categorical	yes	no	n/a	The UnitInfo parameters allow the determination of the "status" of single members within the assessment unit specified by UnitInfo_TU. Possible values of UnitInfo_Id are: - HeadID: the output includes PersonId of the unit's Head. - IsPartner: the output includes a 0/1-variable for being Partner. - IsDependentChild (IsDepChild): the output includes a 0/1-variable for being a dependent child. - IsOwnChild: the output includes a 0/1-variable for being an own child. - IsOwnDependentChild (IsOwnDepChild): the output includes a 0/1-variable for being an own dependent child. - IsDepParent: the output includes a 0/1-variable for being a dependent parent. - IsDepRelative: the output includes a 0/1-variable for being a dependent relative. - IsLoneParent: the output includes a 0/1-variable for being a lone parent.
UnitInfo_TU	taxunit	yes	yes	n/a	Assessment unit for which UnitInfo_Id parameters apply. Note, that outputting unit info variables usually only makes sense if TAX_UNIT is set to an individual assessment unit.
nDecimals	amount	yes	yes	2	Number of decimals of monetary variables to show in output. Values with more decimals are rounded.
Suppress_Void_Message	yes/no	yes	yes	no	If set to yes, the warning for an 'undefined' variable is suppressed.
Replace_Void_By	amount	yes	yes	n/a	Amount to be used for 'undefined' in the output. Note that the default 'void' value is 0.0000000000001.
Append	yes/no	yes	yes	no	If set to yes: any existing content of the output file is removed. If set to no: output is added to any existing content of the output

					file.
MultiplyMonetaryBy	formula	yes	yes	1	All monetary values are multiplied by this factor. (Allows e.g. for the specification of a special exchange rate).

Common Parameters

Name	Type	Optional	Single	Default	Description
Who_Must_Be_Elig	categorical	yes	yes	nobody	Function's calculations are carried out if ... - one (one_member): ... one member of the assessment unit is "eligible" - one_adult: ... one adult member of the assessment unit is "eligible" - all (all_members; taxunit): ... all members of the assessment unit are "eligible" - all_adults: ... all adult members of the assessment unit are eligible - nobody: ... always "eligible" is determined by the variable indicated by parameter Elig_Var
Elig_Var	variable	yes	yes	sel_s	Variable indicating whether a person is "eligible" (see parameter Who_Must_Be_Elig): - zero: person is not eligible - non zero: person is eligible
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.

Summary of parameters for function DefVar

Allows for the definition of intermediate variables.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Var_Dataset	string	yes	yes	n/a	If set, variables are only defined if the respective dataset is used for the run. The wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. be_20*_a?).
Var_SystemYear	string	yes	yes	n/a	If set, variables are only defined if the run concerns the respective system year. The wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. 20??).
[Placeholder]	formula	no	yes	n/a	[Placeholder] stands for the name of the variable, which is defined in the policy column. Optionally a (constant) initial value can be defined in the (respective) system column (i.e. may be system specific). After its definition the variable can (apart from minor specifics) be used in the same way as regular variables.
Var_Monetary	yes/no	yes	yes	yes	If set to no: variable with same group is treated as a non-monetary variable, otherwise as a monetary variable.

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_LimPriority	categorical	yes	yes	upper	Footnote parameter for the further specification of an operand: Possible values: If upper limit (#_UpLim) is smaller than lower limit (#_LowLim) ... - upper: ... upper limit dominates; - lower: ... lower limit dominates; - not defined: ... a warning is issued.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
					Footnote parameter for the further specification of an operand: indicates

#_Amount	amount	yes	yes	n/a	the numeric value of an operand.
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.

Summary of parameters for function DefConst

Allows for the definition of constants.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Const_Dataset	string	yes	yes	n/a	If set, constants are only defined if the respective dataset is used for the run. The wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. be_20*_a?).
Const_SystemYear	string	yes	yes	n/a	If set, constants are only defined if the run concerns the respective system year. The wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. 20??).
[Placeholder]	formula	yes	no	n/a	[Placeholder] stands for the name of the constant, which is defined in the policy column. The value of the constant is defined in the (respective) system column (i.e. may be system specific). Note that the value of the constant can also be (re)defined by e.g. ArithOp, but it must not be household or person specific (i.e. as a constant it must be equal for each individual).
Condition	condition	yes	no	n/a	If set, constant with same group only takes this value if condition is fulfilled (i.e. allows for different values based on conditions). Note that a conditioned constant cannot be used as global value (e.g. in Run_Cond), because it is not equal for all individuals.

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_LimPriority	categorical	yes	yes	upper	Footnote parameter for the further specification of an operand: Possible values: If upper limit (#_UpLim) is smaller than lower limit (#_LowLim) ... - upper: ... upper limit dominates; - lower: ... lower limit dominates; - not defined: ... a warning is issued.
					Footnote parameter for the further specification of an operand:

#_LowLim	formula	yes	yes	-999999999999.99	replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.

Summary of parameters for function Loop

Allows for repeating a part (or all) of the tax-benefit calculations.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Last_Pol	string	no	yes	n/a	Name of loop's last policy.
Last_Func	string	no	yes	n/a	Identifier of loop's last function. The identifier cannot refer to functions which are independent of the policy spine (func_loop, func_unitloop, etc.). (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
Stop_Before_Pol	string	no	yes	n/a	Name of the policy before which the loop ends.
Stop_Before_Func	string	no	yes	n/a	Identifier of the function before which the loop ends. The identifier cannot refer to functions which are independent of the policy spine (func_loop, func_unitloop, etc.). (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
First_Pol	string	no	yes	n/a	Name of loop's first policy.
First_Func	string	no	yes	n/a	Identifier of loop's first function. The identifier cannot refer to functions which are independent of the policy spine (func_loop, func_unitloop, etc.). (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
Start_After_Pol	string	no	yes	n/a	Name of the policy after which the loop starts.
Start_After_Func	string	no	yes	n/a	Identifier of the function after which the loop starts. The identifier cannot refer to functions which are independent of the policy spine (func_loop, func_unitloop, etc.). (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
Loop_Id	string	no	yes	n/a	Unique identifier for the loop. The identifier is used amongst others for constructing the name of the loop count variable 'LoopCount_Loop_Id'.
Num_Iterations	amount	no	yes	n/a	Number of loop iterations.
BreakCond	condition	no	yes	n/a	Break condition checked at the end of the loop. The condition must not contain household/individual specific operators.

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.

Summary of parameters for function UnitLoop

Allows for repeating part (or all) of the tax-benefit calculation.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Last_Pol	string	no	yes	n/a	Name of loop's last policy.
Last_Func	string	no	yes	n/a	Identifier of loop's last function. The identifier cannot refer to functions which are independent of the policy spine (func_loop, func_unitloop, etc.). (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
Stop_Before_Pol	string	no	yes	n/a	Name of the policy before which the loop ends.
Stop_Before_Func	string	no	yes	n/a	Identifier of the function before which the loop ends. The identifier cannot refer to functions which are independent of the policy spine (func_loop, func_unitloop, etc.). (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
First_Pol	string	no	yes	n/a	Name of loop's first policy.
First_Func	string	no	yes	n/a	Identifier of loop's first function. The identifier cannot refer to functions which are independent of the policy spine (func_loop, func_unitloop, etc.). (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
Start_After_Pol	string	no	yes	n/a	Name of the policy after which the loop starts.
Start_After_Func	string	no	yes	n/a	Identifier of the function after which the loop starts. The identifier cannot refer to functions which are independent of the policy spine (func_loop, func_unitloop, etc.). (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
Loop_Id	string	no	yes	n/a	Unique identifier for the loop. The identifier is used amongst others for constructing the name of the loop count variable 'LoopCount_Loop_Id'.
Elig_Unit	taxunit	no	yes	n/a	Calculations are repeated for each Elig_Unit within the household.
Elig_Unit_Cond	condition	yes	yes	{1}	Calculations are only carried out for Elig_Units, which are fulfil this condition.
Elig_Unit_Cond_Who	categorical	yes	yes	all	Defines which members of the unit must fulfil Elig_Unit_Cond to make the unit fulfil the condition. For possible values see the common parameter Who_Must_Be_Elig.
Run_Once_If_No_Elig	yes/no	yes	yes	no	If there is no Elig_Unit within the household, which fulfils Elig_Unit_Cond: If set to yes: policies enclosed by the loop are still carried out (once). If set to no: the polices enclosed by the loop are not carried out.

Common Parameters

Name	Type	Optional	Single	Default	Description
					Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch.

Run_Cond	condition	yes	yes	{1}	Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function Store

Provides a 'save as' functionality for variables, incomelists and components of incomelists.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
PostLoop	string	no	yes	n/a	Copies of stored variables are called VarName_PostLoopIteration. For example: Var = yem, PostLoop = myloop: name of copies = yem_myloop1 (1. iteration), yem_myloop2 (2. iteration), etc.
PostFix	string	no	yes	n/a	Copies of stored variables are called VarName_PostFix. For example: Var = yem, PostFix = bkup: name of copy = yem_bkup.
Var	variable	yes	yes	n/a	Name of a variable to be stored.
Var_Level	taxunit	yes	yes	n/a	Alternative assessment unit: variable's value is assessed based on this unit. Note that the parameter is only relevant if PostLoop refers to a UnitLoop.
IL	incomelist	yes	yes	n/a	Incomelist whose entries (i.e. variables) are to be stored.
IL_Level	taxunit	yes	yes	n/a	Alternative assessment unit: the value of the variables included in IL is assessed based on this unit. Note that the parameter is only relevant if PostLoop refers to a UnitLoop.
Level	taxunit	yes	yes	n/a	Alternative assessment unit: the value of all variables is assessed based on this unit, except for those which have a specific own Var_Level/IL_Level. Note that the parameter is only relevant if PostLoop refers to a UnitLoop.

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.

Summary of parameters for function Restore

Sets variables back to some previous value (stored by function Store).

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
PostLoop	string	no	yes	n/a	copy of stored variable is called varname_postfix e.g. var1 = yem, postfix = bkup: name of copy = yem_bkup
PostFix	string	no	yes	n/a	copy of stored variable is called varname_postloopiteration e.g. var1 = yem, postloop = lp: name of copies = yem_lp1 (1. iteration), yem_lp2 (2. iteration), etc.
VoidSim	yes/no	yes	yes	n/a	yes: all simulated variables are set to undefined (VOID)
Keep_SimVar	variable	yes	no	n/a	to be used with parameter voidsim: the indicated variables are not set to undefined (but keep their value)
Iteration	amount	yes	yes	most recent	to be used with parameter postloop: variables are set back to the value they had when function store was carried out in the indicated iteration

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.

Summary of parameters for function ChangeParam

Allows for changing the values of parameters of other functions.

Note: to change the switches of policies or functions use ChangeSwitch.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Param_Id	string	no	per group	n/a	Identifier for the parameter to change.
Param_NewVal	string	no	per group	n/a	New value for parameter (to be exchanged at read time).
Dataset	string	yes	no	n/a	If any 'Dataset' parameter is used, the change only takes place if one of them matches the dataset of the concerned run. The wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. be_20*_a?).

Summary of parameters for function ChangeSwitch

Allows for changing the switches of policies or functions.

This function is only available with EM3.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
PolFun	string	no	per group	n/a	The name or identifier of the policy respectively the identifier or symbolic-identifier (add-ons) of the function. (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
SwitchNewVal	string	no	per group	n/a	New switch of the policy/function.

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.

Summary of parameters for function Totals

Allows for the calculation of aggregates of variables or incomelists over the whole population or a selected subgroup.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Varname_Sum	string	yes	yes	n/a	Name for variable to store population sum of Agg_IL / Agg_Var. (e.g. if Varname_Sum = $sum and Agg_Var = yem: sum is stored in $sum_yem)
Varname_Mean	string	yes	yes	n/a	Name for variable to store mean of Agg_IL / Agg_Var. (e.g. if Varname_Mean = $mean and Agg_Var = yem: mean is stored in $mean_yem)
Varname_Median	string	yes	yes	n/a	Name for variable to store median of Agg_IL / Agg_Var. (e.g. if Varname_Median = $median and Agg_Var = yem: median is stored in $median_yem)
Varname_Decile	string	yes	yes	n/a	Name for variable to store decile points of Agg_IL / Agg_Var. (e.g. if Varname_Decile = $dec and Agg_Var = yem: decile points are stored in $dec1_yem, ..., $dec9_yem)
Varname_Quintile	string	yes	yes	n/a	Name for variable to store quintile points of Agg_IL / Agg_Var. (e.g. if Varname_Quintile = $quint and Agg_Var = yem: decile points are stored in $quint1_yem, ..., $quint4_yem)
Varname_Count	string	yes	yes	n/a	Name for variable to store Agg_IL / Agg_Var's count of non-zero values. (e.g. if Varname_Count = $count and Agg_Var = yem: count is stored in $count_yem)
Varname_PosCount	string	yes	yes	n/a	Name for variable to store Agg_IL / Agg_Var's count of positive values. (e.g. if Varname_PosCount = $poscnt and Agg_Var = yem, count is stored in $poscnt_yem)
Varname_NegCount	string	yes	yes	n/a	Name for variable to store Agg_IL / Agg_Var's count of negative values. (e.g. if Varname_NegCount = $negcnt and Agg_Var = yem: count is stored in $negcnt_yem)
Varname_Min	string	yes	yes	n/a	Name for variable to store smallest value of Agg_IL / Agg_Var. (e.g. if Varname_Min = $min and Agg_Var = yem: minimum is stored in $min_yem)
Varname_Max	string	yes	yes	n/a	Name for variable to store largest value of Agg_IL / Agg_Var. (e.g. if Varname_Max = $max and Agg_Var = yem: maximum is stored in $max_yem)
Agg_IL	incomelist	yes	no	n/a	Incomelist for which to calculate aggregates.
Agg_Var	variable	yes	no	n/a	Variable for which to calculate aggregates.
Incl_Cond	condition	yes	yes	{1}	Condition that must be fulfilled by the assessment unit, as defined by TAX_UNIT, to be taken into account by the statistic.
Incl_Cond_Who	categorical	yes	yes	all	Defines which members of the assessment unit must fulfil Incl_Cond to make the unit fulfil the condition. For possible values see the common parameter Who_Must_Be_Elig.
Use_Weights	yes/no	yes	yes	yes	If set to yes weights are used to calculate the aggregates.
Weight_Var	variable	yes	yes	dwt	Specifies an alternative weight variable.

Common Parameters

Name	Type	Optional	Single	Default	Description
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function DropUnit

Allows for dropping individuals, families or households with special characteristics from the calculations.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Drop_Cond	condition	no	yes	n/a	Condition defining which assessment units (i.e. indvidual, families or households) are to be dropped.
Drop_Cond_Who	categorical	yes	yes	one	Defines which members of the assessment unit must fulfil Drop_Cond to make the unit fulfil the condition. For possible values see the common parameter Who_Must_Be_Elig.

Common Parameters

Name	Type	Optional	Single	Default	Description
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.

Summary of parameters for function KeepUnit

Allows for keeping only individuals, families or households with special characteristics within the calculations.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Keep_Cond	condition	no	yes	n/a	Condition defining which assessment units (i.e. indvidual, families or households) are to be kept.
Keep_Cond_Who	categorical	yes	yes	one	Defines which members of the assessment unit must fulfil Keep_Cond to make the unit fulfil the condition. For possible values see the common parameter Who_Must_Be_Elig.

Common Parameters

Name	Type	Optional	Single	Default	Description
TAX_UNIT	taxunit	no	yes	n/a	Assessment unit for function's calculations.
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function IlVarOp

Allows for operations on the content (i.e. the variables) of an incomelist.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Operand	formula	no	yes	n/a	Variables of Operator_IL are multiplied/increased by operand. Example: Operator_IL = ils_earns = { yem / yse } Operand = 2.5 All other parameters are set to their defaults. Result: yem = yem * 2.5, yse = yse * 2.5
Operand_Factors	yes/no	no	yes	no	Variables of Operator_IL are multiplied/increased by factors of Operator_IL. Example: Operator_IL = ils_earns_mul = { 3 yem / -2 yse } Operand_Factors = yes All other parameters are set to their defaults. Result: yem = yem * 3, yse = yse * (-2)
Operand_IL	formula	no	yes	n/a	Variables of Operator_IL are multiplied/increased by variables of Operand_IL. Example: Operator_IL = ils_earns = { yem / yse } Operand_IL = il_multiply = { poa / yiy } All other parameters are set to their defaults. Result: yem = yem * poa, yse = yse * yiy
Operator_IL	incomelist	no	yes	n/a	Incomelist containing the variables on which the operation takes place.
Operation	categorical	yes	yes	MUL	Possible values: - MUL: variables of Operator_IL are multiplied by Operand. - ADD: variables of Operator_IL are increased by Operand. - NEGTOZERO: negative variables of Operator_IL are set zo zero, postive variables keep their value.
Sel_Var	categorical	yes	yes	ALL	Possible values: ALL: operation takes place on all variables of Operator_IL. MAX: operation takes place on the highest variable of Operator_IL. MIN: operation takes place on the smallest variable of Operator_IL. MINPOS: operation takes place on the smallest positive variable of Operator_IL (no operation occurs, if there is no positive variable).

Common Parameters

Name	Type	Optional	Single	Default	Description
Who_Must_Be_Elig	categorical	yes	yes	nobody	Function's calculations are carried out if ... - one (one_member): ... one member of the assessment unit is "eligible" - one_adult: ... one adult member of the assessment unit is "eligible" - all (all_members; taxunit): ... all members of the assessment unit are "eligible"

					- all_adults: ... all adult members of the assessment unit are eligible - nobody: ... always "eligible" is determined by the variable indicated by parameter Elig_Var
Elig_Var	variable	yes	yes	sel_s	Variable indicating whether a person is "eligible" (see parameter Who_Must_Be_Elig): - zero: person is not eligible - non zero: person is eligible
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function RandSeed

Sets the starting point for generating a series of pseudorandom numbers.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Seed	amount	yes	yes	1	Integer value as starting point for random number generation.

Common Parameters

Name	Type	Optional	Single	Default	Description

Summary of parameters for function CallProgramme

Allows for calling an

external application.
Note that the function is only available under Windows as it uses platform specific code.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Programme	string	no	yes	n/a	Name of the application to be called. Example: Excel.exe
Path	string	yes	yes	n/a	Path to the application. Dispensable if the application is installed at the standard path for programmes. However, note that the path is also used as the working directory for the application. Example: C:\Program Files\Microsoft Office\ Note that spaces can be used without encapsulating the path by ".
Argument	string	yes	no	n/a	Programme argument to be passed to the application. Example: C:\EuromodFiles\Tools\EMT_FillTemplate.xls (to e.g. open Excel with this workbook)
UnifySlash	yes/no	yes	yes	yes	If set to yes all occurrences of / are replaced by \ in parameters Path and Argument.
Wait	yes/no	yes	yes	no	If set to yes: EUROMOD calculations are stopped until the called programme terminates. If set to no: EUROMOD calculations continue without waiting.
RepByEMPath	string	yes	yes	..\	Any occurrences of RepByEMPath in parameters Path, and Argument are replaced by the path of the current EUROMOD content (see EUROMOD Installation and Architecture). Example: RepByEMPath = &, Argument = &Tools\SomeFile.xls, EUROMOD content is stored at C:\EuromodFiles\ Result: Argument is interpreted as C:\EuromodFiles\Tools\SomeFile.xls. Remark: Technically the programme obtains knowledge about the path of the EUROMOD content via the parameter EMCONTENTPATH in the configuration file (see EUROMOD Installation and Architecture - EUROMOD software (user interface and executable) - The configuration file).

Common Parameters

Name	Type	Optional	Single	Default	Description

Summary of parameters for function DefInput

Allows for reading

values for one or more EUROMOD variables from a text file.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
path	string	no	yes	n/a	Path of the input file.
file	string	no	yes	n/a	Name of the input file.
RowMergeVar	variable	no	yes	n/a	ColMergeVar != n/a means 'lookup mode', i.e. one EUROMOD variable (per person) is set to a specific value looked up in the input file. The variable is defined by InputVar. Which value is looked up depends on the person's value of the variable defined by RowMergeVar as well as the person's value of the variable defined by ColMergeVar. The lookup value is the value of the cell whose row and column headers in the input file coincide with these variables. ColMergeVar = n/a: means 'input mode', i.e. one or more EUROMOD variables (per person) are set to values defined in a specific row of the input file. The header row of the input file names the respective variables, while the other rows contain their values in the same order. One of these variables must be the variable defined by RowMergeVar. The row selected for a person is the one where the person's value of RowMergeVar coincides with the value of this variable in the input file. Note that, other than for the 'lookup mode', the RowMergeVar does not need to be located in the first column of the input file. Note that the variables inputted in 'input mode' as well as the variable defined by InputVar must exist, i.e. be defined in the variables file or by EUROMOD functions (e.g. DefVar).
ColMergeVar	variable	yes	yes	n/a	See description of parameter RowMergeVar.
InputVar	variable	yes	yes	n/a	See description of parameter RowMergeVar.
DefaultIfNoMatch	amount	yes	yes	n/a	Input variable(s) are set to this value if no match can be established. If not defined an error message is issued if no match can be established.
IgnoreNRows	amount	yes	yes	0	The first n rows of the input file are ignored (e.g. because they contain descriptions or headings).
IgnoreNCols	amount	yes	yes	0	The first n columns of the input file are ignored (e.g. because they contain descriptions or headings).
DecSepComma	yes/no	yes	yes	no	If set to yes, input file is assumed to use comma as decimal separator otherwise point.

DoRanges	yes/no	yes	yes	no	If set to yes, matches are established by considering RowMergeVar and ColMergeVar in the input file as upper limits. Example: RowMergeVar dag in the input file set to 10, 30, 70, 150 Interpretation: all persons aged up to 10 (including) are assigned the value of the 10-column, all persons aged older than 10 up to 30 are assigned the value of the 30-column, all persons aged older than 30 up to 70 are assigned the value of the 70-column, all persons aged older than 70 up to 150 are assigned the value of the 150-column, no match is found for anyone older than 150.
MultiSystemUse	yes/no	yes	yes	yes	If set to yes, the content of the input file is kept in memory until the programme terminates. If set to no, the respective memory is released once contents are assigned to EUROMOD variables. Potential reasons for setting to no: used by only one system and keeping memory usage small; content of file changes between uses of different systems. Potential reason for setting to yes: used by several systems and keeping reading times low.
RepByEMPath	string	yes	yes	n/a	Any occurrences of RepByEMPath in parameter Path are replaced by the path of the current EUROMOD content (see EUROMOD Installation and Architecture). Example: RepByEMPath = &, Path = &Tools\SomeFile.xls, EUROMOD content is stored at C:\EuromodFiles\ Result: Path is interpreted as C:\EuromodFiles\Tools\SomeFile.xls. Note that this parameter is only available under Windows as it uses platform specific functions. Remark: Technically the programme obtains knowledge about the path of the EUROMOD content via the parameter EMCONTENTPATH in the configuration file (see EUROMOD Installation and Architecture - EUROMOD software (user interface and executable) - The configuration file).

Common Parameters

Name	Type	Optional	Single	Default	Description

Summary of parameters for function Scale

Allows for scaling monetary variables and monetary parameters.

This function is only available with EM3.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
FactorVariables	amount	yes	yes	1	All monetary variables are multiplied by this factor. The operation concerns all monetary variables existing at the time of execution (i.e. used before). If the function is part of a loop, the scaling is done in each iteration.
FactorParameter	amount	yes	yes	1	All monetary parameters are multiplied by this factor. Monetary parameters are identified by having a period (#m, #y, etc). Rates (#mr, #yr, etc.) are considered non-monetary. The operation concerns all parameters defined after the Scale function. Loops have no effect (i.e. the scaling is done once). Note that the factor is always applied on the original parameter values. Thus a subsequent Scale with e.g. FactorParameter 0.5 after a Scale with FactorParameter 0.3 would not result in a factor 0.15, but 0.5 is applied. Consequently a FactorParameter 1 undoes all scaling.

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.

Summary of parameters for function AddHHMembers

Allows for adding persons to households.

This function is only available with EM3.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Add_Who	categorical	no	yes	n/a	Defines whether children or partners or other persons are added. Possible values: 'Child', 'Partner', 'Other'.
ParentCond	condition	no	yes	n/a	Condition a person must fulfill to be a parent (i.e. a child is added for the person). Only relevant if Add_Who=Child.
PartnerCond	condition	no	yes	n/a	Condition a person must fulfill to be a partner (i.e. a partner is added for the person). Only relevant if Add_Who=Partner.
HHCond	condition	no	yes	n/a	Condition the household must fulfill for adding a new person. Only relevant if Add_Who=Other.
IsPartnerParent	yes/no	yes	yes	yes	If yes, the partner of the new parent (person fulfilling ParentCond) gets the other partent of the added child. Only relevant if Add_Who=Child.
[Placeholder]	formula	yes	no	n/a	[Placeholder] stands for the name of the variable, which is defined in the policy column. Variable's initial value for the new person is set in system-column.
FlagVar	variable	yes	yes	n/a	If indicated, variable is set to 1 for persons added by this AddHHMembes function. Note that the variable must exist. Also note that the variable is not changed for any other persons (thus it can e.g. be used to hold the flag for more than one AddHHMembers functions).

Common Parameters

Name	Type	Optional	Single	Default	Description
Run_Cond	condition	yes	yes	{1}	Function is only carried out if the condition is fulfilled. The parameter is intended to be a conditional switch. Thus the condition must not be individual or household based, but refer to a specific processing state or other global condition.
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_LowLim	formula	yes	yes	-999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is smaller.
#_UpLim	formula	yes	yes	999999999999.99	Footnote parameter for the further specification of an operand: replaces operand if operand is higher.
#_Amount	amount	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates the numeric value of an operand.
#_Level	taxunit	yes	yes	n/a	Footnote parameter for the further specification of an operand: indicates an alternative assessment unit for an operand.
#_AgeMin	amount	yes	yes	-999999999999	Parameter of several queries (e.g. nDepChildrenInTu).
#_DataBasename	string	yes	yes	n/a	Parameter of query IsUsedDatabase.
#_AgeMax	amount	yes	yes	999999999999	Parameter of several queries (e.g. nDepChildrenInTu).

#_N	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_M	amount	yes	yes	n/a	Parameter of query IsNtoMchild.
#_Val	variable/incomelist	yes	yes	n/a	Parameter of query HasMaxValInTu.
#_Income	variable/incomelist	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerIncome).
#_Info	variable	yes	yes	n/a	Parameter of several queries (e.g. GetPartnerInfo).
#_Unique	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.
#_Adults_Only	yes/no	yes	yes	no	Parameter of query HasMaxValInTu.

Summary of parameters for function Break

Allows the user to break the run at any point inside the spine.

This function is only available with EM3.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
ProduceOutput	yes/no	yes	yes	yes	If yes, all variables and incomelists are dumped in an output file. Note that dumping incomelists is likely to lead to warnings ("... use of not initialised variable ...").
OutputFileName	string	yes	yes	name of first output-file after the Break	Specifies the name of the output file. If not indicated or n/a the program searches for the first DefOutput after the Break and takes the name from there (parameter File). The path is defined as usual for DefOutput (i.e. via the User Interface respectively the configuration file).
ProduceTUinfo	yes/no	yes	yes	no	If true, TU info is dumped to the output file for each TU defined before the Break (see DefOuput UnitInfo-parameters).

Common Parameters

Name	Type	Optional	Single	Default	Description

Summary of parameters for function AddOn_Applic

Add-on function: Provides a short description of the add-on and specifies for which systems the add-on is applicable.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Description	string	no	yes	n/a	A short description of the add-on (amongst others to be displayed by the run-tool).
Sys	string	yes	no	n/a	Name of (a) system(s) for which the add-on is applicable. Wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. *_200?).
SysNA	string	yes	no	n/a	Name of (a) system(s) for which the add-on is not applicable. Note that not applicability prevails over applicability (i.e. SysNA is stronger than Sys). Wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. ??_2005).

Common Parameters

Name	Type	Optional	Single	Default	Description

Summary of parameters for function AddOn_Pol

Add-on function: Allows for adding a policy.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Pol_Name	string	no	yes	n/a	Name of the policy to insert (must be unique).
Insert_Before_Pol	string	no	yes	n/a	Name of the policy before which the add-on policy should be inserted.
Insert_After_Pol	string	no	yes	n/a	Name of the policy after which the add-on policy should be inserted.
Allow_Duplicates	yes/no	yes	yes	n/a	If set to no: adding a policy twice (i.e. Pol_Name identical) leads to an error message. If set to yes: adding a policy twice works, but "references" cannot be used, i.e. the usage of functions like ChangeParam and Loop is not possible.

Common Parameters

Name	Type	Optional	Single	Default	Description

Summary of parameters for function AddOn_Func

Add-on function: Allows for adding a function.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Id_Func	string	no	yes	n/a	Identifier of the function to insert. (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
Insert_Before_Func	string	no	yes	n/a	Identifier of the function before which the add-on function should be inserted. (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
Insert_After_Func	string	no	yes	n/a	Identifier of the function after which the add-on function should be inserted. (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)

Common Parameters

Name	Type	Optional	Single	Default	Description

Summary of parameters for function AddOn_Par

Add-on function: Allows for adding parameters.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Insert_Func	string	no	yes	n/a	Identifier of the function into which the parameter should be inserted. (See 'EUROMOD Functions - Identifiers and the placeholders =cc= and =sys=' for information on identifiers.)
[Placeholder]	string	yes	no	n/a	[Placeholder] stands for the name of the parameter to insert, defined in the policy column. The value of the parameter is specified in the respective system column (i.e. the may be system specific).

Common Parameters

Name	Type	Optional	Single	Default	Description

Summary of parameters for function AddOn_ExtensionSwitch

Add-on function: Allows add-ons to switch extensions on or off.

This function is only available with EM3.

Function Specific Parameters

Name	Type	Optional	Single	Default	Description
Extension_Name	string	no	no	n/a	Name of the extension (long- and short name can be used) e.g. 'benefit take-up adjustments' or 'BTA'.
Extension_Id	string	no	no	n/a	Identifier of the extension (look up in XML or use Extension_Name instead).
Extension_Switch	yes/no	no	no	n/a	Whether this extension should be turned on or off.
Dataset	string	yes	no	n/a	If any 'Dataset' parameter is used (several can be used within one group), the setting is only applied if one of them matches the dataset of the concerned run. The wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. be_20*_a?).
System	string	yes	no	n/a	If any 'System' parameter is used (several can be used within one group), the setting is only applied if one of them matches the system of the concerned run. The wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. be_20??).

Common Parameters

Name	Type	Optional	Single	Default	Description

Queries

Query	Description	Parameters	Aliases
IsHeadOfTu	Returns 1 if a person is the 'Head' of the assessment unit, i.e. fulfils the fixed head condition and the ExtHeadCond of the assessment unit specification, 0 otherwise. See the (summary) description of function DefTU for further details.		IsHead
IsPartner	Returns 1 if a person is the 'Partner' of the assessment unit, i.e. fulfils the PartnerCond of the assessment unit specification, 0 otherwise. See the (summary) description of function DefTU for further details.		IsPartnerOfHeadOfTu
IsDepChild	Returns 1 if a person is a 'dependent child', i.e. fulfils the DepChildCond of the assessment unit specification, 0 otherwise. See the (summary) description of function DefTU for further details.		IsDependentChild
IsOwnChild	Returns 1 if a person is an 'own child', i.e. fulfils the OwnChildCond of the assessment unit specification, 0 otherwise. See the (summary) description of function DefTU for further details.		
IsOwnDepChild	Returns 1 if a person is an 'own dependent child', i.e. fulfils the OwnDepChildCond of the assessment unit specification, 0 otherwise. See the (summary) description of function DefTU for further details.		IsOwnDependentChild
IsLooseDepChild	Returns 1 if a person is a 'loose dependent child', i.e. fulfils the LooseDepChildCond of the assessment unit specification, 0 otherwise. See the (summary) description of function DefTU for further details.		IsLooseDependentChild
	Returns 1 if a person is a 'dependent parent', i.e.		

IsDepParent	fulfils the DepParentCond of the assessment unit specification, 0 otherwise. See the (summary) description of function DefTU for further details.		IsDependentParent IsDepPar
IsDepRelative	Returns 1 if a person is a 'dependent relative', i.e. fulfils the DepRelativeCond of the assessment unit specification, 0 otherwise. See the (summary) description of function DefTU for further details.		IsDependentRelative IsDepRel
IsLoneParentOfDepChild	Returns 1 if a person is a 'lone parent', i.e. fulfils the LoneParentCond of the assessment unit specification, 0 otherwise. See the (summary) description of function DefTU for further details.		IsSingleParentOfDepChild
IsMarried	Returns 1 if a person is married, i.e. variable dms = 2, 0 otherwise.		
IsCohabiting	Returns 1 if a person has a partner, i.e. variable idpartner > 0, and is not married, i.e. variable dms != 2, 0 otherwise.		
IsWithPartner	Returns 1 if a person has a partner, i.e. variable idpartner > 0, irrespective of being married, i.e. the variable dms, 0 otherwise.		
IsInEducation	Returns 1 if a person is in education, i.e. variable dec > 0 and variable les = 6, 0 otherwise.		
IsDisabled	Returns 1 if a person is disabled, i.e. variable ddi > 0. If ddi does not exist, ddilv is used and a respective warning is issued. If neither ddi nor ddilv exist the query is ignored (person is considered not disabled) and a respective warning issued.		
IsCivilServant	Returns 1 if a person is a civil servant, i.e. variable lcs = 1, 0 otherwise		
	Returns 1 if a person is a blue-collar worker, i.e. variable loc = 6 (skilled agriculture) or 7 (craft worker) or 8		

IsBlueColl	(plant operator) or 9 (elementary occupation) or 0 (armed force), 0 otherwise.		
IsParent	Returns 1 if this person or its partner has at least one child, i.e. idMother[child] = idPerson[person] or idFather[child] = idPerson[person] or idMother[child] = idPartner[person] or idFather[child] = idPartner[person], 0 otherwise. Note that the query is independent of any assessment unit definition (e.g. as there is no child definition, 'children' have no age limit).		
IsParentOfDepChild	Returns 1 if: ● either at least one 'own dependant child' of the person or its partner belongs to the assessment unit ● or at least one 'loose dependent child' belongs to the assessment unit and the person is 'Head' or 'Partner' where: ● for being an 'own dependent child' query IsDepChild must apply ● for being a 'loose dependent child' query IsLooseDepChild must apply ● for being 'Head' query IsHead must apply ● for being 'Partner' query IsPartner must apply 0 otherwise.		
IsLoneParent	Is equivalent to IsParent & !IsWithPartner.		IsLonePar IsSingleParent IsSinglePar
nChildrenOfCouple#x	Returns the number of a couple's children. For a specific assessment unit 'couple's children' must fulfil each of the following three conditions: 1. idMother[child] = idPerson[Head] or idMother[child] = idPartner[Head] or idFather[child] = idPerson[Head] or idFather[child] = idPartner[Head] 2. dag[child] >= parameter #_AgeMin 3. dag[child] <= parameter #_AgeMax	#_AgeMin; optional #_AgeMax; optional	nChOfCouple

	where ● for being 'Head' query IsHead must apply ● the head's 'Partner' does not have to be in the same assessment unit Note that children of the whole household are counted (not just children belonging to the assessment unit) and that no assessment unit specific child definition is applied.		
nDepChildrenOfCouple#x	Returns the number of a couple's dependent children. For a specific assessment unit 'couple's dependent children' must fulfil each of the following five conditions: 1. query IsDepChild or query IsLooseDepChild applies 2. child is part of the assessment unit 3. idMother[child] = idPerson[Head] or idMother[child] = idPerson[Partner] or idFather[child] = idPerson[Head] or idFather[child] = idPerson[Partner] or idMother[child] = idFather[child] = 0 4. dag[child] >= parameter #_AgeMin 5. dag[child] <= parameter #_AgeMax where: ● for being 'Head' query IsHead must apply ● for being 'Partner' query IsPartner must apply	#_AgeMin; optional #_AgeMax; optional	nDepChOfCouple
nPersInUnit#x	Returns the number of persons in the assessment unit who fulfil dag >= parameter #_AgeMin and dag<=#_AgeMax.	#_AgeMin; optional #_AgeMax; optional	npersonsintu nPersonsInTaxunit nPersInTu nPersTaxunit
nAdultsInTu#x	Returns the number of adults in the assessment unit who fulfil dag >= parameter #_AgeMin and dag <= parameter #_AgeMax. For being counted as adult the query IsDepChild must not apply.	#_AgeMin; optional #_AgeMax; optional	nAdultsInTaxunit
	Returns the number of dependent children in the		

nDepChildrenInTu#x	assessment unit who fulfil dag >= parameter #_AgeMin and dag <= parameter #_AgeMax. For being counted as dependent child the query IsDepChild must apply.	#_AgeMin; optional #_AgeMax; optional	nDepChInTu nDepChildrenInTaxunit nDepChInTaxunit
nLooseDepChildrenInTu	Returns the number of loose dependent children in the assessment unit. For being counted as loose dependent child the query IsLooseDepChild must apply.		nLooseDepChInTu nLooseDepChildrenInTaxunit nLooseDepchInTaxunit
nDepParentsInTu	Returns the number of dependent parents in the assessment unit. For being counted as dependent parent the query IsDepParent must apply.		
nDepRelativesInTu	Returns the number of dependent relatives in the assessment unit. For being counted as dependent relative the query IsDepRelative must apply.		
nDepParentsAndRelativesInTu	Is equivalent to nDepParentsInTu + nDepRelativesInTu.		
IsNtoMchild#x	Returns 1 if a person belongs to the n to m oldest dependent children of the assessment unit, 0 otherwise. For being counted as 'dependent child' query IsDepChild must apply. n and m are defined by parameters #_N and #_M. If children are equally aged, they are sorted by idPerson, i.e. children with lower idPerson are considered to be older than their equally aged siblings. Example: #_N = 2, #_M=4. Assesment unit comprises 6 children aged 12, 10, 8, 6, 4 and 2. Condition is fulfilled for the 2nd-oldest (10), the 3rd-oldest (8) and the 4th-oldest (6) child.	#_M; compulsory #_N; compulsory	
	Returns 1 if a person has the highest value of a specific variable in the assessment unit, 0 otherwise. Parameter #_val defines the specific value. Parameter #_unique treats the case of several persons with the same highest		

HasMaxValInTu#x	value: If set to yes, the return value is 1 for the person first occurring in the dataset only, otherwise for all persons with this value. If parameter #_adults_only is set to yes dependent children cannot be selected unless there is no adult in the assessment unit. For being regarded a 'dependent child' query IsDepChild must apply.	#_val; compulsory #_unique; optional #_adults_only; optional	IsRichestInTu
HasMinValInTu#x	Returns 1 if a person has the lowest value of a specific variable in the assessment unit, 0 otherwise. Parameter #_val defines the specific value. Parameter #_unique treats the case of several persons with the same lowest value: If set to yes, the return value is 1 for the person first occurring in the dataset only, otherwise for all persons with this value. If parameter #_adults_only is set to yes dependent children cannot be selected unless there is no adult in the assessment unit. For being regarded a 'dependent child' query IsDepChild must apply.	#_val; compulsory #_unique; optional #_adults_only; optional	
GetPartnerIncome#x	Returns the income of a person's partner. The partner is defined as the person in household whose 'idperson' equals the concerned person's 'idpartner'. Parameter #_income defines the relevant income (as variable or incomelist). Note that the query is independent of any assessment unit definition (and being married). Also note that the query returns zero if there is no partner. Finally note that the query has an alias 'GetPartnerInfo' which is supposed to be used when a non-monetary information upon the partner is requested (e.g. age). There is however no real difference (i.e. GetPartnerIncome and GetPartnerInfo return the same result).	#_income; compulsory	
GetCoupleIncome#x	Returns the income of a person and her/his partner. The partner is defined as the person in household whose 'idperson' equals the concerned person's 'idpartner'. Parameter #_income defines the relevant income (as variable or incomelist). Note that the query is independent of any assessment unit definition (and being married).	#_income; compulsory	
	Returns the income of a person's (both) parents. The parents are defined as the persons in		

GetParentsIncome#x	household whose 'idperson' equals the concerned person's 'idfather' or 'idmother'. Parameter #_income defines the relevant income (as variable or incomelist). Note that the query is independent of any assessment unit definition. Also note that the query returns zero if there are no parents, respectively the income of one parent if there is only one.	#_income; compulsory	
GetMotherIncome#x	Returns the income of a person's mother. The mother is defined as the person in household whose 'idperson' equals the concerned person's 'idmother'. Parameter #_income defines the relevant income (as variable or incomelist). Note that the query is independent of any assessment unit definition. Also note that the query returns zero if there is no mother. Finally note that the query has an alias 'GetMotherInfo' which is supposed to be used when a non-monetary information upon the mother is requested (e.g. age). There is however no real difference (i.e. GetMotherIncome and GetMotherInfo return the same result).	#_income; compulsory	
GetFatherIncome#x	Returns the income of a person's father. The father is defined as the person in household whose 'idperson' equals the concerned person's 'idfather'. Parameter #_income defines the relevant income (as variable or incomelist). Note that the query is independent of any assessment unit definition. Also note that the query returns zero if there is no father. Finally note that the query has an alias 'GetFatherInfo' which is supposed to be used when a non-monetary information upon the father is requested (e.g. age). There is however no real difference (i.e. GetFatherIncome and GetFatherInfo return the same result).	#_income; compulsory	
GetOwnChildrenIncome#x	Returns the income of a person's children (all of them). The children are defined as persons in household whose 'idfather' or 'idmother' equals the concerned person's 'idperson'. Parameter #_income defines the relevant income (as variable or incomelist). Note that the query is independent of any assessment unit definition. Also note that the query returns the sum of all children's income and zero if there are no children.	#_income; compulsory	
GetSystemYear	The query tries to extract the system's year from the system's name, by searching for 4 subsequent digits, and returns the result if found, -1 otherwise.		

GetDataIncomeYear	Returns the income year of the applied dataset.		
IsOutputCurrencyEuro	Returns 1 if the output currency of the system is Euro, 0 if national currency.		
IsParamCurrencyEuro	Returns 1 if the system is parametrised in Euro, 0 if in national currency.		
GetExchangeRate	Returns system's exchange rate from euro to national currency.		
IsUsedDatabase#x	Returns 1 if a specific database is the database used by the current run. Parameter #_DataBasename defines the respective database. The wildcards * and ? can be used, where * stands for any character and ? stands for one arbitrary character (e.g. be_20*_v?).	#_DataBasename; compulsory	

Change log

This section provides a log of the changes to the EUROMOD help pages and is intended to keep EUROMOD users informed about changes to the model or the description from one version to the other.

EUROMOD Installation and Architecture

EUROMOD Architecture can be best described by dividing it into "software" and "content".

- EUROMOD software basically comprises the EUROMOD user interface and the EUROMOD executable. For a detailed description see EUROMOD Installation and Architecture - EUROMOD software (user interface and executable).
- EUROMOD content is a specific file structure. The structure is mainly composed by the EUROMOD parameter files, which essentially describe how the tax-benefit systems of the countries included in EUROMOD are implemented in the model. For a detailed description see EUROMOD Installation and Architecture - EUROMOD content (parameter files).

When users launch EUROMOD they open in fact the EUROMOD user interface which takes care of essentially the whole communication between users and the model. The two main tasks of the user interface are to allow for

- viewing and modifying the implementation of the tax-benefit systems of the counties included in EUROMOD
- running the model

For accomplishing the former task, the user interface reads and modifies information stored in the EUROMOD content. Essentially it reads from and writes to parameter files in XML format, which contain the instructions the model needs for representing the countries’ tax-benefit systems. To do so the user interface needs to know where to find the EUROMOD content, that means it requires information where the respective files structure is stored. This information is indicated and can be changed via the main menu's item *Open project* (see Working with EUROMOD - Open project). Note that in principle it is possible that more than one EUROMOD content exists. In other words there are two (or more) such file structures (which may contain different versions of the parameter files). In this case the dialog can be used to "switch" between these versions. Also note that it is possible to simultaneously launch two (or more) instances of the user interface, which may display different EUROMOD contents.

The user interface stores the EUROMOD content it points to when it is closed and thus, on reopening, is able to display the same content. Also with the installation of a more recent version of the software this information will not get lost. Thus, only when EUROMOD is installed for the very first time, the user interface does not have any information which content to display and therefore opens the configuration dialog to allow users to select a respective path. For more information see EUROMOD Installation and Architecture - Installing EUROMOD.

For accomplishing the latter task – running the model - the user interface starts the EUROMOD executable, which in fact carries out the tax-benefit calculations and produces respective output in text format. To do so the executable needs two sets of information. Firstly, it reads the XML parameter files as stored in the EUROMOD content (and possibly modified by the user interface). Secondly, it reads a configuration file produced by the user interface for this specific run. Most importantly the configuration file tells the executable which tax-benefit systems to run on which data.

Installing EUROMOD

Prerequisites

- Windows XP, Windows Vista, Windows 7 or Windows 8
- At least 150 MB free disc space
- Administrator permissions and internet connection (as the Setup will download and install Microsoft .NET framework files)

Installation or Upgrade of EUROMOD software

see EUROMOD Installation and Architecture - EUROMOD software (user interface and executable)

Download the EUROMOD software installation file *Setup_EMSoftware_X.Y.exe*, start it and follow the instructions. This checks and, if necessary, downloads and installs .NET Framework. It also adds a shortcut to the Start menu and to the desktop, and adds EUROMOD to the installed programmes in Windows Control Panel.

Specification the EUROMOD user interface's "references"

Setup automatically starts the EUROMOD user interface. Upon the first installation of the interface, it opens the open-project-dialog, which allows for selecting the project (i.e. the specific versions of country-implementations) the user interface refers to (see Working with EUROMOD - Open project, in particular paragraph *Opening the user interface for the very first time*).

You may also want to configure other important settings, as for example where the model, as a default, generates output or where the model, as a default, looks for input data. This purpose is served by the project configuration dialog, opened via the item *Project Configuration* of the main menu.[1] See Working with EUROMOD - Configure project for a detailed description.

The user interface maintains these specifications. Thus, on a subsequent start of the user interface, and also on a subsequent installation, it does not request this information anymore.

[1] You find the main menu above the *Run EUROMOD* button. Press the little arrow to open it.

EUROMOD software (user interface and executable)

EUROMOD software essentially consists of the EUROMOD user interface and the EUROMOD executable. Concerning the tasks of these programmes please see EUROMOD Installation and Architecture. Moreover, EUROMOD software comprises some additional programmes and files required by the model: most importantly the programme that provides built-in help (EUROMODHelp.chm) and some configuration files. All these parts are installed, respectively updated by the EUROMOD installation programme (see EUROMOD Installation and Architecture - Installing EUROMOD).

Technically the user interface is implemented using Microsoft Visual C# and requires a Microsoft Windows environment. The executable is a C++ programme and, apart from some special functions, can principally be compiled to run on other environments as well. However, this option is currently not utilised nor are there plans to check it out in the near future. Also in principle the executable can be run independently from the user interface. For this a respective configuration file must be set up, which is also the way the user interface communicates with the executable.

In fact, the communication between the user interface and the executable takes place as follows:

From the user interface to the executable: The user interface launches the executable and passes information via a configuration file. The most important information passed is, where the EUROMOD content (i.e. XML parameter files) is stored and which tax-benefit-systems are to be run. For a detailed description and the storage place of the configuration file see EUROMOD Installation and Architecture - EUROMOD software (user interface and executable) - The configuration file.

Note that one run of the executable refers to one specific dataset. That means, if users select several countries for run, respectively tax-benefit systems of one country which are to be run with different datasets, the user interface launches a corresponding number of instances of the executable with a respective number of configuration files (compare Working with EUROMOD - Running EUROMOD).

From the executable to the user interface: The executable reports about its progress via the standard output stream, whereas it writes errors and warnings to the standard output stream for errors. The user interface keeps "listening" to all instances of the executable it started by reading from these streams. Thus the user interface is able to show progress and display error logs (compare Working with EUROMOD - Running EUROMOD). Apart from reporting warnings and errors via the standard output stream for errors, the executable generates a respective error-log-file to provide for more permanent information.

The configuration file

The EUROMOD configuration file is a file that is generated by default from the Run Dialogue every time you run the model. It contains all the information the executable needs to perform the run and was in the past the only way to perform a run from command line. It is still required if you are running the "old executable" (i.e. EM2) from command line. The new EM3 executable can still use the configuration file, but it also provides an easier way to run the model from command line, through the use of command line parameters. The syntax to run euromod with a configuration file is:

EM2: "*<executable path>\EUROMOD.exe*" "*<configuration file path>\configuration file name.xml*"

EM3: "*<executable path>\EM_ExecutableCaller.exe*"

"*<configuration file path>\configuration file name.xml*"

Usually the configuration file is generated by the EUROMOD user interface and stored in the *Temp* folder of the EUROMOD content (see EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Organisation of files). The user interface names the configuration file *EMConfigGUID.xml*, where *GUID* stands for **G**lobally **U**nique **ID**entifier and is a unique reference number formed by a 32-character hexadecimal string. For the user interface it is necessary to use unique names for configuration files as it frequently starts more than one run of the executable at the same time.

The content of the configuration file is composed by the

following information. *xxx* denotes an example.

Content of configuration file	Comments
<?xml version="1.0" standalone="true"?>	
<EMConfig>	
<COUNTRY_FILE>*SL.xml*</COUNTRY_FILE>	names of the parameter country to run (compare I Installation and Architecture - E (parameter files) - Format of c files)
<DATACONFIG_FILE>*SL_DataConfig.xml*</DATACONFIG_FILE>	
<PARAMPATH>*C:\Euromod\EuromodFiles\XMLParam\Countries\SL\*</PARAMPATH>	folder where (the two above) p stored

<OUTPUTPATH>C:\Euromod\EuromodFiles\output\</OUTPUTPATH>	folder to store model output (c with EUROMOD - Running paragraph *Selecting the o*
<DATAPATH>C:\Euromod\EuromodFiles\input\</DATAPATH>	folder where input-dataset is s Working with EUROMOD -
<DATASET_ID>CBA7E428-F8E4-4CEB-8A5E-9ACE73987DD7</DATASET_ID>	unique ID of the input-datas cc_DataConfig.xml (compar Installation and Architecture - EU (parameter files) - Format of co files)
<SYSTEM_ID>4027fd02-1691-4216-a3e2-f00b980bb06f</SYSTEM_ID>	unique IDs of the systems to run (compare EUROMOD Installatic - EUROMOD content (paramete country parameter
<SYSTEM_ID>6b06872b-6a65-4750-869c-9aa41c7cf6e9</SYSTEM_ID>	
<STARTHH>-1</STARTHH>	ID, as stored in input-dataset, of to be included in the calculatic lower/upper limit (compare EUROMOD - Running EURO *Limiting the output to selecte*
<LASTHH>-1</LASTHH>	
<DECSIGN_PARAM>.</DECSIGN_PARAM>	Decimal sign used in paramete Working with EUROMOD - A countries - Handling of the deci separators in EURO
<EMVERSION>F6.36</EMVERSION>	compare Working with EUROM
<UIVERSION>Beta2</UIVERSION>	see Working with EUROMOD
<ERRLOG_FILE>C:\Euromod\EuromodFiles\output\201306041351_errlog.txt</ERRLOG_FILE>	name and path of the file to log e (see Working with EUROMO Finding errors
<LOG_WARNINGS>yes</LOG_WARNINGS>	setting of option *Do not stop on* (see Working with EUROM EUROMOD, paragraph *Adv*
<LOG_RUNTIME>no</LOG_RUNTIME>	setting of option *Log runtime in c* with EUROMOD - Running paragraph *Advanced s*
<HEADER_DATE>201306041351</HEADER_DATE>	date to be used as prefix for EI Working with EUROMOD - Run paragraph *Running the selecte combinations*)
<OUTFILE_DATE>-</OUTFILE_DATE>	optional date to be used as prefix files, corresponds to setting of c *output-filename* (see Working w Running EUROMOD, parag *settings*)
<CONFIGPATH>C:\Euromod\EuromodFiles\XMLParam\Config\</CONFIGPATH>	folder where the EUROMOD var (compare EUROMOD Installatic - EUROMOD content (paramete the variables fil
<EMCONTENTPATH>C:\Euromod\EuromodFiles\</EMCONTENTPATH>	folder containing the EUROMOI EUROMOD Installation and EUROMOD content (param
<POLICY_SWITCH>yem_??=4027fd02-1691-4216-a3e2-f00b980bb06f=off</POLICY_SWITCH>	instructions for the executable to of switchable policies, i *SwitchablePolicy_SearchPattern* for more information see W EUROMOD - Changing Cou Administrating policy
<POLICY_SWITCH>yem_??=6b06872b-6a65-4750-869c-9aa41c7cf6e9=on</POLICY_SWITCH>	
<LAST_RUN>yes</LAST_RUN>	used by the user interface to inc run is the last of a series (instruct

	close the error log file (with ar
<ISPUBLICVERSION>*no*</ISPUBLICVERSION>	informs the executable if the rur EUROMOD public version (se EUROMOD - Generating a EU version)
</EMConfig>	

[1] Assuming that the EUROMOD file structure is always organised as described in EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Organisation of files it would be enough for the executable to know the EMCONTENTPATH, as PARAMPATH and CONFIGPATH can be deduced from EMCONTENTPATH. In fact, the reason for having three parameters is historical. The reason for still keeping them is twofold.

Firstly, and more essentially, add-ons generate temporary parameter files, which are not read from the usual path, but from the *Temp* folder. Secondly, the executable does not need to know about the organisation of files, i.e. no redundancy of information is necessary.

Command line parameters

There are two ways to run the EUROMOD executable from command line, and both can use either a configuration file or command line parameters to define the run properties:

1. With a set of named parameters **(this is the recommended way)**
2. With a set of unnamed parameters

Running EUROMOD with a set of named parameters

This is now the preferred method of calling the executable from command line, as it provides the golden ballance between user-friendliness and flexibility. The syntax for calling the executable with a set of named parameters is:

EM_ExecutableCaller <parameters>

In this case, the order in which you have to place the parameters is arbitrary. The currently available list of parameters is described in the table below:

Parameter	Usage	Comments
-config <*config_path*>	Special	The path to a configuration file (e.g. -config "<*configuration file path*>\configuration file name.xml"). If this parameter is specified, then there is no need to specify any other parameters, as everything else can be specified within the configuration file. You can read more about the configuration file in the relevant help section. **The usage for the remaining parameters below assume that the -config parameter is not specified.**
-emPath <*project_path*>	Required	The path to the EUROMOD project you want to run (e.g. -emPath "C:\Euromod\EuromodFiles_I1.0+").
-sys <*system_name*>	Required	The name of the system you wish to run (e.g. -sys "BE_2014").
-data <*dataset_name*>	Required	The name of the dataset you wish to use (e.g. -data "BE_2015_a1").
-outPath <*output_path*>	Optional	The folder where you want to write the model output (e.g. -outPath "C:\Euromod\EuromodFiles_I1.0+\Output").
-dataPath <*data_path*>	Optional	The folder where you store your input data (e.g. -inPath "C:\Euromod\EuromodFiles_I1.0+\Input").
-addOn <*addOn_name*>|<*add-on-system*>	Optional	This allows you to run an add-on (e.g. -addOn "MTR|MTR_BE"). Note that one can use to parameter for adding as many add-ons as desired.
-extSwitch <*extension_name*>=<*value*>	Optional	This allows you to change an extension value (e.g. -extSwitch "BTA_??=off"). Note that one can use the parameter for as many extension value changes as required.
-globalPath <*global_path*>	Optional	The folder where you store your global files (e.g. -globalPath "C:\Euromod\EuromodFiles_I1.0+\XMLParam\Config\"). It is useful in cases where you need to store different versions of your country files in different places (e.g. because you are creating reforms from Stata), but your golbal files do not change and you do not want to have to replicate the full project folder structure for each reform.

-forceOutputInEuro	Optional	This parameter forces the EUROMOD executable to produce all output in Euro.
-forceSequentialRun	Optional	This parameter forces the EUROMOD executable to use only a single thread to run the model (by default, EUROMOD tries to fully utilise your CPU by using all available threads).
-forceSequentialOutput	Optional	This parameter forces the EUROMOD executable to use only a single thread to prepare the output file (while everything else still runs in multiple threads). It can be useful in cases where RAM is sparse, as it cuts memory usage to almost half, while the total execution time is still much faster than when using -forceSequentialRun.

Some examples for illustration:

"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" -config "C:\Euromod\EuromodFiles_I1.0+\XMLParam\Temp\EMConfig30aab8ef-ae5b-4601-842c-fdeb67c49116.xml"

"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" -emPath "C:\Euromod\EuromodFiles_I1.0+" -sys BE_2014 -data BE_2015_a1

(or *"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" -data BE_2015_a1 -sys BE_2014 -emPath "C:\Euromod\EuromodFiles_I1.0+")*

(or *"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" -sys BE_2014 -emPath "C:\Euromod\EuromodFiles_I1.0+" -data BE_2015_a1)*

"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" -emPath "C:\Euromod\EuromodFiles_I1.0+" -sys BE_2014 -data BE_2015_a1 -dataPath "C:\Euromod\ModelData" -outPath "C:\Euromod\OutputFolder"

"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" -emPath "C:\Euromod\EuromodFiles_I1.0+" -sys BE_2014 -data BE_2015_a1 -addOn "MTR|MTR_BE"

"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" -emPath "C:\Euromod\EuromodFiles_I1.0+" -sys BE_2014 -data BE_2015_a1 -globalPath "C:\Euromod\EuromodFiles_I1.0+\XMLParam\Config" -forceOutputInEuro

(Note: quotes are only necessary where an argument (e.g. path) contains blanks but it is important to ***not*** end your path with a backslash because this confuses windows!)

Running EUROMOD with a set of unnamed parameters

This simplified way of calling the executable from command line is the old way of running EUROMOD and in this case the **order of parameters is crucial**. Please note that this method is still available for backwards compatibility only, but will likely be phased out in the future so we do not recommend using it.

If you provide the EUROMOD executable with a **single unnamed parameter**, then this is expected to be the path to a configuration file. The syntax for this is:

EM_ExecutableCaller <config_file>

For example:

"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" "C:\Euromod\EuromodFiles_I1.0+\XMLParam\Temp\EMConfig30aab8ef-ae5b-4601-842c-fdeb67c49116.xml"

If you provide the EUROMOD executable with **three to five unnamed parameters**, then the syntax becomes:

EM_ExecutableCaller <project_path> <system_name> <data_name> [<data_path>] [<output_path>]
where the default value for <data_path> is "<project_path>\Input"
and the default value for <output_path> is "<project_path>\Output"

Some examples for illustration:
"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" "C:\Euromod\EuromodFiles_I1.0+" BE_2014 BE_2015_a1
"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" "C:\Euromod\EuromodFiles_I1.0+" BE_2014 BE_2015_a1 "C:\Euromod\ModelData"
"C:\Program Files\EUROMOD\Executable\EM_ExecutableCaller.exe" "C:\Euromod\EuromodFiles_I1.0+" BE_2014 BE_2015_a1 "C:\Euromod\ModelData" "C:\Euromod\OutputFolder"
(Note: quotes are only necessary where an argument (e.g. path) contains blanks but it is important to ***not*** end your path with a backslash because this confuses windows!)

Note that the dataset indicated with *data_name* must be registered via the *Configure Databases* dialog. This is necessary for obtaining important data settings like the income year of the data.

EUROMOD content (parameter files)

As stated in EUROMOD Installation

and Architecture, EUROMOD content is a specific file structure, which contains the information required by the EUROMOD software to accomplish its tasks (see EUROMOD Installation and Architecture - EUROMOD

software (user interface and executable)). The concrete content of the structure is described in EUROMOD

Installation and Architecture - EUROMOD content (parameter files) - Organisation of files, while its main content, the EUROMOD parameter files, are described in EUROMOD Installation and Architecture -

EUROMOD content (parameter files) - Format of country parameter files.

Finally EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Format of the variables file describes a special parameter file, which contains information about EUROMOD variables.

Organisation of files

The EUROMOD file structure, also referred to as "content" (in contrast to "software") is organised as follows:

Folder / sub folder	**Content**
XMLParam	This folder contains the files where the executable and the user interface draw their information from.
Countries	This folder contains the EUROMOD parameter files, i.e. the files that contain the implementation of the countries' tax-benefit systems in EUROMOD. There is one sub folder for each country. The name of this folder is the short name of the country (e.g. DE for Germany, HU for Hungary, EL for Greece, ...). Each country's folder contains two XML-files (cc stands for the country's short name): ***cc.xml*** contains the parameters that describe the country's tax-benefit system (and some other information). ***cc_DataConfig.xml*** contains information on the input datasets that are used to simulate these tax-benefit systems. For the composition of these files see EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Format of country parameter files. Moreover, the country's folder contains an image in portable network graphic format (png) used by the user interface to display the country's flag.[1]
AddOns	This folder contains special EUROMOD parameter files, which contain the implementation of EUROMOD add-ons. Similar to countries, there is one sub folder per add-on. The name of the folder is the short name of the add-on (e.g. MTR.xls for the Marginal Tax Rate add-on). Each add-on's folder holds one XML-file containing the parameters that describe the add-on's tasks. The structure of the file is in principle the same as for a country's *cc.xml*. See EUROMOD Installation and Architecture - EUROMOD content (parameter files) - Format of country parameter files for its composition. Moreover, the add-on's folder contains an image in portable network graphic format (png) used by the user interface to display the add-on's symbol. Also see EUROMOD Functions - EUROMOD add-ons and the special functions AddOn_Applic, AddOn_Pol, AddOn_Func and AddOnPar for more information on implementing add-ons and Working with EUROMOD - Running EUROMOD for information on the application of add-ons.[2]
Config	This folder contains two files: ***VarConfig.xml*** stores the list of EUROMOD variables in XML-format - see Working with EUROMOD - Administration of EUROMOD variables. ***EuromodVersion.txt*** stores the EUROMOD version number – see Working with EUROMOD - Open project, paragraph *EUROMOD Version*. Moreover, the folder contains a subfolder ***Images***, which contains two images in portable network graphic format (png) used by the user interface as a default to display a country's flag or an add-on's symbol, if it does not possess an own flag/symbol.
Temp	The folder contains temporary files, which can be deleted without consequence.
Input	This folder may contain input data - see Working with EUROMOD - Open project.
Output	This folder may be used as default output folder - see Working with EUROMOD - Open project.
Applications	This folder contains external tools - see Working with EUROMOD - Applications.

[1] In addition the folder may temporary contain a small text file, which contains information on who uses the country currently (see Working with EUROMOD - Saving and file locking).

[2] In addition the folder may temporary contain a small text file, which contains information on who uses the add-on currently (see Working with EUROMOD - Saving and file locking).

Format of country parameter files

The information required by EUROMOD for implementing a country's tax-benefit system is stored in two XML-files (cc stands for the country's short name):

cc.xml (e.g. HU.xml) essentially contains the parameters that describe the country's tax-benefit system.

cc_DataConfig.xml (e.g. HU_DataConfig.xml) contains information on the input datasets that are used to simulate these tax-benefit systems.

The paragraphs below describe the XML-structure of these files, where *xxx* denotes an example and ***xxx*** denotes a comment. Note that these files are used as well by the user interface as by the EUROMOD executable.

cc.xml

```
<?xml version="1.0" ?>
<CountryConfig xmlns="http://euromod.com/CountryConfig.xsd">

    <Country>

    <ID>863BDA5C-077E-4508-86E9-0A7F69812A1C</ID>
<Name>Simpleland</Name>

    <ShortName>sl</ShortName>

    for each implemented tax benefit system:

    <System>

    <ID>250851CB-3685-4844-B98D-0BA4C3854808</ID>
<CountryID>863BDA5C-077E-4508-86E9-0A7F69812A1C</CountryID>
<Name>SL_demo</Name>

    <CurrencyOutput>euro</CurrencyOutput>

    <CurrencyParam>euro</CurrencyParam>

    <ExchangeRateEuro>1</ExchangeRateEuro>

    <HeadDefInc>ils_origy</HeadDefInc>

    <Private>no</Private>

    <Order>1</Order>
```

for each policy of the tax benefit system:

<Policy>

<ID>*374DA5B0-8E08-4664-96B2-BC1794E5AEBC*</ID> <SystemID>*250851CB-3685-4844-B98D-0BA4C3854808*</SystemID>

<ReferencePolID></ReferencePolID>

<Name>*Uprate_sl*</Name> <Type>*def*</Type> <Comment>*<![CDATA[DEF: UPRATING FACTORS]]>*</Comment> <Private>*no*</Private> <PrivateComment></PrivateComment>

<Switch>*on*</Switch> <Order>*1*</Order> *for each function of the policy:*

<Function>

<ID>*4F1388B1-4D49-4CE0-BF32-C17328206B73*</ID> <PolicyID>*374DA5B0-8E08-4664-96B2-BC1794E5AEBC*</PolicyID> <Name>*Uprate*</Name> <Comment>*<![CDATA[]]>*</Comment> <Private>*no*</Private> <PrivateComment></PrivateComment>

<Switch>*on*</Switch> <Order>*1*</Order> *for each parameter of the function:*

<Parameter>

<ID>*427912DF-0EC6-4785-B0A3-0E322C08EA28*</ID> <FunctionID>*4F1388B1-4D49-4CE0-BF32-C17328206B73*</FunctionID> <Name>*def_factor*</Name> <Comment>*<![CDATA[]]*></Comment> <PrivateComment></PrivateComment>

<Value>*1*</Value> <ValueType>*amount*</ValueType> <Order>*2*</Order> <Group></Group>

<Private>*no*</Private>

</Parameter>

</Function>

</Policy>

</System>

</Country>

for each conditional format (information required by the user interface - see Working with EUROMOD - Conditional formatting):

<ConditionalFormat>

<ID>*68649e4e-50dd-4460-bac7-b4db9695abd7*</ID> <BackColor></BackColor> <ForeColor>*FF80FF00*</ForeColor/> <Condition>*{*#m*}*</Condition> <BaseSystemName> </BaseSystemName> ***for each system concerned by the conditional format:***

<ConditionalFormat_Systems>

<ConditionalFormatID>*68649e4e-50dd-4460-bac7-b4db9695abd7*</ConditionalFormatID> <SystemName>*SL_demo*</SystemName> **</ConditionalFormat_Systems>**

<ConditionalFormat>

</CountryConfig>

cc_DataConfig.xml

<?xml version="1.0" ?>

<DataConfig xmlns="http://euromod.com/DataConfig.xsd">

for each available dataset:

<DataBase>

<ID>*DDBC477B-91E6-4A60-BF93-A08C538D57FF*</ID>

<Name>*sl_demo_v4*</Name>

<Comment>*<![CDATA[]]>*</Comment> <YearCollection></YearCollection>

<YearInc></YearInc>

<Currency>*euro*</Currency> <FilePath></FilePath>

<DecimalSign>.</DecimalSign> <Private></Private>

<UseCommonDefault></UseCommonDefault>

</DataBase>

for each system-dataset combination (see EUROMOD Basic Concepts - EUROMOD input and output):

<DBSystemConfig>

<SystemID>*250851CB-3685-4844-B98D-0BA4C3854808*</SystemID> <SystemName>*SL_demo*</SystemName> <DataBaseID>*DDBC477B-91E6-4A60-BF93-A08C538D57FF*</DataBaseID> <UseDefault></UseDefault>

<UseCommonDefault></UseCommonDefault> ***outdated, to be removed once not used anymore***

<Uprate></Uprate>

<BestMatch>*yes*</BestMatch> ***for each default of a switchable policy (see Working with EUROMOD - Changing Countries' Setting - Administrating policy switches):***

<PolicySwitch>

<SwitchablePolicyID>*79513cab-d092-477d-b17e-442882767cbf*</SwitchablePolicyID> <SystemID>*250851CB-3685-4844-B98D-0BA4C3854808*</SystemID> <DatabaseID>*DDBC477B-91E6-4A60-BF93-A08C538D57FF*</DatabaseID> <Value>*<![CDATA[on]]>*</Value> **</PolicySwitch>**

</DBSystemConfig>

</DataConfig>

Format of the variables file

The file storing EUROMOD variables (see Working with EUROMOD - Administration of EUROMOD variables) has the following format (*xxx* denotes an example and ***xxx*** denotes a comment):

<?xml version="1.0" ?>
<VarConfig xmlns="http://euromod.com/VarConfig.xsd">

for each variable:

<Variable>

<ID>*8B0EEE12-6D72-42C6-9730-1A195AB98AA7*</ID> <Name>*<![CDATA[afc]]>*</Name> <Monetary>*0*</Monetary>

<AutoLabel>*<![CDATA[assets : financial capital]]>*</AutoLabel> ***for each country specific description:***

<CountryLabel>

<ID>*6CCEF1AC-34A6-49F0-9D71-D219FAE6A90A*</ID>
<VariableID>*8B0EEE12-6D72-42C6-9730-1A195AB98AA7*</VariableID>
<Country>*at*</Country>

<Label>*<![CDATA[Finanzkapital]]>*</Label> **</CountryLabel>**

</Variable>

for each acronym type:

<AcronymType>

<ID>*D0E67181-46DF-4629-83DB-4115448365FE*</ID> <LongName>*<![CDATA[demographic]]>*</LongName> <ShortName>*<![CDATA[d]]>*</ShortName> ***for each acronym level:***

<AcronymLevel>

<ID>*6CCA8A22-0FDD-4A25-BFEC-3EDB56F435CC*</ID>
<Index>*1*</Index>

<TypeID>*D0E67181-46DF-4629-83DB-4115448365FE*</TypeID> <Name>*<![CDATA[main]]>*</Name> ***for each acronym:***

<Acronym>

<ID>*FB3DBC6E-9238-4BAA-BF36-0E0171EA0A52*</ID>

<LevelID>*6CCA8A22-0FDD-4A25-BFEC-3EDB56F435CC*</LevelID> <Name>*<![CDATA[ct]]>*</Name> <Description>*<![CDATA[country]]>*</Description> ***for each category:***

<Category>

<AcronymID>*FB3DBC6E-9238-4BAA-BF36-0E0171EA0A52*</AcronymID> <Value>*1*</Value>

<Description>*<![CDATA[at]]>*</Description> **</Category>**

</Acronym>

</AcronymLevel>

</AcronymType>

for each switchable policy:

<SwitchablePolicy>

<ID>*79513cab-d092-477d-b17e-442882767cbf*</ID> <NamePattern>*<![CDATA[yem_??]]>*</NamePattern> <LongName>*<![CDATA[Minimum Wage]]>*</LongName> **</SwitchablePolicy>**

</VarConfig>

EUROMOD Version Control

The help for EUROMOD version control is split in two parts

- **Workflow guide & basic concepts** and
- **VC forms & functions**

The **Workflow guide & basic concepts** pages, in principle, can be read like a textbook. They instruct users in a twofold way: Firstly, they provide them with the knowledge they need to understand EUROMOD version control. Secondly, they give how-to-do instructions on the operations necessary for Version Control. The work flow instructions are arranged bottom-up. That means they start with instructions for general usage, i.e. tasks most users will need to know and/or will operate with higher frequency. This is followed by instructions for less frequent use and by less users (e.g. administrative tasks).

The **VC forms & functions** are of more technical

nature. They concentrate on describing the technical procedure and sometimes provide technical background. Moreover, they are more comprehensive than the work flow instructions which sometimes concentrate on the most usual

applications. The VC forms & functions pages are from time to time referenced in the work flow instructions, so that the latter can concentrate on "what's going on" while the former explain "how that's accomplished".

Obviously reading this part of the help like a textbook does not make sense, rather it is accessed via a search request or via the work flow instructions.

Workflow guide & basic concepts

This section provides instructions for the most common workflows Version Control (VC) may be used for. However, to better understand these workflows, you need to first understand the basic concepts of VC.

VC is an online repository for EUROMOD projects, that can store multiple versions of each project, allowing users to better track and manage their model's development. In these help pages, we refer to the projects stored in VC as *VC projects*. Every VC project is composed of several *Units*. A *Unit* can be a country, an Add-on, a configuration file (such as the Variables file or the Exchange Rates file), or a special folder (like the Log folder or the Input folder).

As already mentioned, VC can store multiple versions of each project. These versions are called *Bundles*. Unit versions refer to the bundle they were first committed. If you create a new bundle with a specific unit updated, but with other units unchanged, the other units will not change their version number and in fact, technically, they will not even be uploaded again in VC. So this way, a specific unit version can exist in multiple bundles.

The unit level is the finest level of control VC has. This is true both in terms of allowing the user to download specific units only, but also in the aspect that users can have unit-specific access rights. So a user may have both read & write access for given unit, while having only read rights for another unit and no rights at all for a third (which would mean that this unit is completely invisible to the user).

In the following pages, you will learn how to handle the most common workflows in VC:

- Connecting to a VC project
- Downloading a VC project bundle
- Creating a new bundle for an existing VC project
- Administrating VC projects

Downloading a project bundle

On regular basis, the EUROMOD team releases updates of the EUROMOD core project,

which always include new or improved implementations of countries' tax-benefit systems, frequently new or improved implementations of EUROMOD add-ons and sometimes even new countries. This section describes how to download a given bundle, either as a clean new project or to update your existing local EUROMOD project. The following instructions apply to any VC project, including ones that you have created yourself.

Creating a new local project

In order to create a new (clean) local project out of an online bundle, you

first need to click the *New Project* button to bring up the *New Project* form.

You can find this button in two places: in the main menu and in the Version Control ribbon within the *PROJECT* group.

There you should check the *Project on VC* option. Note that if you were

not already logged into the VC, you will be asked to login now. Then you need to choose the Project and the Bundle you want to download. As with any new project,

you also need to specify the *Project Folder* and *Project Name* at

the top of the form. If you do not want to get the full bundle, you can specify which units you want to download by clicking on the *Define Content* button.

So for example, if you want to create a new project out of the latest available

master version, you need to:

- Click on *New Project* button to bring up the *New Project* form.
- Set the new project path and name.
- Check the *Project on VC* checkbox.
- Select the *EUROMOD_MASTER_VERSION* on the first combo-box.
- And finally click on OK.

The latest bundle will be selected by default, and all the available units will be included. Please note that it may take some time to download the whole project.

Similarly, if you want to create a new project for a specific paper that focuses for example on AT, BE, DE & ES, and you want it based on master version H0.45, then you need to:

- Click on the *New Project* button to bring up the *New Project* form.
- Set the new project path and name.
- Check the *Project on VC* checkbox.
- Select the *EUROMOD_MASTER_VERSION* on the first combo-box.
- Select H0.45 on the second combo-box.
- Click on the *Define Content* button.
- In the *Define Project Content* form, chose only the units you want

  to download (AT, BE, DE, ES, all config files, possibly any Add-ons you need as well as the Input & Log folders).
- Click OK on the *Define Project Content* form.
- And finally click OK on the *New Project* form.

Updating an existing local project

Sometimes however, you are already working with an existing local project and you need to bring it up-to-date with a given online bundle, but you want to do so without losing any of your local changes.

You can achieve this by following the steps below:

- Log into the VC, by clicking the *Click to Log In* button on the VC ribbon.
- Click on the *Download Bundle* on the VC ribbon to bring up the

Version Control - Download Bundle form. If you were already connected to a project, it will be preselected in the *Project* combo-box.

- Select the Project and Bundle you want to download. At this point, VC will download detailed information for each unit and compare them to your local units. This may take some time.
- Once done, the grid will be filled with detailed information. For each unit, you will have the option to either keep your version, overwrite it with the online version, or get merge support so that you can use the *Merge Tool* to combine the two versions. Note however, that some options may not be available for all units. For example, the Merge Tool does not support some unit types, so the *Get Merge Support*

  option will be disabled. Also, if a local unit is already matching the contents of the online unit, no further action can be taken.
- Click on the *Download* button. VC will not touch any units for which

  you selected *Keep Local Version*, it will overwrite all units where you

  selected *Get Online Version* and will create a merge environment for all

  units where you selected *Get Merge Support*.
- If you selected *Get Merge Support* for one or more Countries or Add-ons,

  you can now open them in the UI, click the *Merge Country/Add-on* button on

  the VC ribbon and then click *Yes* to open the Merge Tool using the downloaded merging environment.
- If you selected *Get Merge Support* for the *VARCONFIG* unit

  (aka the Variables file), you can now click the *Merge Variables* button on

  the VC ribbon and then click *Yes* to open the Merge Tool using the downloaded merging environment.

For further information on these issues, please see Download Bundle.

Creating a new bundle for an existing VC project

If you are working on a given project and you have it on Version Control (VC), you will regularly find the need to save your work by uploading it to VC. You can do this by creating and uploading a new *Bundle*. This section describes how you can connect to a given project and how to create a new bundle for it.

Connect your local project to a VC project

To create a new bundle for a given VC project, you need to first connect your local project (that has the changes you want to upload) to the VC project. The first step in doing this is to login to Version Control. You can do this by going to the VC ribbon, clicking on the *Click to Log In* button and entering your details. Once you are logged in, the *Click to Connect* button in the VC ribbon will become enabled. Clicking on it will present you with a list of VC projects for which you have at least read access. Select the project you want to update and click on *OK*.

Merge your changes into the VC project

In order to create a new bundle, you need to start a *Merge* by clicking on the *Start Merging* button. Note that only one person may have an active *Merge* for any given VC project. Once you connect to the VC project, you will get a notification if anyone else is currently merging, or if you have a pending merge of your own. If you successfully start a new merge, the VC will send an email to everyone involved in this project, informing them you did so. A similar email is sent out once you *finish* or *abort* your merge.

So once your new merge has started, the first thing you need to make sure is that you have the latest changes that exist online, to avoid overwriting any changes that someone else may have committed in the meantime. You can do this by using the Download Bundle button, and selecting to *merge* your current changes with the latest unit versions available online.

When this is done and you have the model ready for upload, you need to click on the *Finish Merging* button. This will open the *Version Control - Upload Bundle* form that will allow you to select which units you want to update. Any units you check will be overwritten by your local units in the new bundle, and any units you leave unchecked will not be affected (the new bundle will include the unit versions included in the latest online bundle). Note that VC will automatically generate the new bundle version number, but you can also change this manually if you want to. To complete your merge, just click on the *Upload* button.

For further information on these issues, please see Start/Finish Merging.

Administrating VC projects

From time to time, you will have a specific research project to complete that requires you to create a separate EUROMOD project. For example, you may need to write a comparative paper for 10 countries. For cases like these, you can create and manage your own custom VC projects and this section shows you how to do this.

Create new VC project

Creating a new empty VC project is very easy and can be completed in three simple steps:

- *Log in*
- Click the *Add VC Project* button
- Select a name for this project and click OK

The above steps will generate a new empty VC project and connect your local project to the new VC project.

Administrating the content of your VC project

Once your new VC project is ready and you are connected to it, you will need to add some content (units) to it. You can do this by clicking the *Administrate Content* button in the *ADMIN* section of the VC ribbon to bring up the *Version Control - Administrate Content* form. This form contains a listbox with all the available version controlled units for the VC project you are connected to. Using the buttons at the bottom of the form, you can:

- Add Local Units to VC: which allows you to upload your existing local units into the VC project.
- Remove Units from VC: which will remove the selected units from VC (including all the versions of these units ever committed!).
- Get Units from VC: which will create a local copy of the selected VC units (taking the latest version available).

Assuming this is a new VC project you just created, you would need to click the *Add Local Units to VC* button, select the units you want to include in the VC project and type a short message that will be bound to the first commit of these units. Note that this will not automatically generate a bundle for your project, so you would still need to do this manually using the Start Merging button.

Administrating the Users of your VC project

So now you have a VC project that is ready to be used. The next step will be to allow other people to use it. When you create a new VC project, you are automatically added as the administrator of this project. To add or remove other users, or to change their access rights, you need to click the *Administrate Users* button, which will bring up the *Version Control - Administrate Project Users* form. Here you will see a list of all existing users, along with their access rights for this project. You can add further users by clicking on the *Add Users* button on the top right. This will give you a list of all the VC users (who are not already part of this project) and allow you to give them access to this project.[1] Furthermore, you can finetune the type of access each user has, down to the unit level. The highest access a user can have is to be an *Admin* for this project, meaning that (s)he has the power to change everything related to this project or even to remove this project from VC completely. The *Default Right* is the right that the user has by default for all units, unless otherwise specified. Clicking the *Refine Rights* button will let you specify the unit-level access for the specific user, where:

- *Upload*: means the user can see, upload & download the specific unit
- *Download*: means the user can see & download the specific unit
- *None*: means the user can not even see that the specific unit exists

Removing Bundles

From time to time you may find yourself in a situation (e.g. due to a human error) where you need to remove an already committed bundle. Assuming that you are already logged in, connected to a project and that you have admin rights, you can do this by clicking the *Remove Bundle* button to bring up the *Version Control - Remove Bundles* form. There you can see a list of all the available bundles. To remove one, simply select it and click the *Remove Bundle(s)* button. Note that removing the bundle will not actually remove the committed units -

they will just no longer be connected to a specific bundle, so unless you change your local files, they will appear as *Pending* the next time you try to merge (if you do not change your local version in the meantime).

Removing a VC project

If you are an *Admin* for a VC project, then you are also able to completely remove it from Version Control. You can do this by clicking the *Remove VC Project* button to bring up the *Version Control - Remove Project* form. Then you can simply select the project you want to remove and click *OK*. After confirming your choice, the VC project you selected will be removed.

[1] If the user you would like to add is not in that list, then you need to contact the VC responsible in the core EUROMOD team so that (s)he can add this user to the Version Control platform.

VC forms & functions

This section provides detailed & specialised information for every form and function in Version Control. In the following pages, you can find more information about:

- Logging in and out
- Identification Settings
- Connecting & Disconnecting a project
- New Project
- Download Bundle
- Start/Finish Merging
- The Merge Tool
- The Merge Tool for Variables
- Administrate Content
- Administrate Users
- Remove Bundles
- Add VC Project
- Remove VC Project

Logging in and out

Logging in

If the user is currently not logged in, the login-button in the ribbon *Version Control* shows a closed door and the text *Click to Log In*. Clicking the button opens the login-dialog, which asks for a *User Name* and *Password*. If these are correctly provided, the user is logged in, which is visualised by the button now showing an open door (and the text *Click to Log Out*).

Forgot password?

If you click the question-mark button alongside the *Password* field, you are posed a 'forgot password question', provided that you have already indicated a (correct) *User Name* and that such a question is defined in your identification settings (see Identification Settings). If you answer the question correctly, the user interface tells you the forgotten password.

Logging out

If the user is currently logged in, the login-button in the ribbon *Version Control* shows an open door and the text *Click to Log Out*. Clicking the button will log the user out and "close the door", i.e. the button now shows a closed door (and the text *Click to Log In*). Moreover, all other buttons (except for the Merge Tool) in the ribbon are deactivated, i.e. no version control operations are available until the user logs back in.

Button activation and user rights

If a button in the ribbon *Version Control* is activated, that does not automatically mean, that the operation is available. More precisely, the user interface does not take user rights into account for (de)activating the buttons. Thus it is possible that a user is e.g. able to press the *Upload* button, but then gets an information, that she has no right to upload the country.

Identification Settings

The button *Identification* in the Version Control ribbon's section *USER* opens a dialog which allows for changing the current user's identification settings.

Within the *Version Control - Identification Settings* form, you can see and change (most of) your personal information. The only thing you cannot change is your *User Name* as this is used to track your actions and identify you within VC. The *Name* and *Last Name* are just used for displaying information purposes. The *E-Mail* address is actually used to send you notifications whenever someone changes projects you are part of, so please make sure this is correct.

The button *Change Password* allows you to change your password settings. Clicking it prompts the request to "... indicate your current password ...". Correctly doing so opens a dialog which allows you to indicate a *New Password*. Before you confirm this new password with *OK*, please retype it in the respective field, to make sure that no spelling mistake happened.

The dialog also allows you to specify a *Forgot Password Question* and a respective *Forgot Password Answer*. This information is used in the Version Control - Login dialog to retrieve your password, in case you forget it. Obviously, it is advisable to pose a question for which the answer is likely to be known only by you.

Connecting & Disconnecting a project

Connecting a project

To connect your local project to the version control system, click the button *Click to Connect* in the in the *Version Control* ribbon's section *PROJECT*. The *Click to Connect* button is only available if the project is not yet connected to version control, as it changes to *Click to Disconnect* when a connection is already established - see below. Furthermore, this button is only enabled if you are already *logged in*.

Clicking on the Connect button will bring up the *Connect Project to Version Control* form. There you will see a listing of all projects which are available to you[1] on the version control system. Ticking the respective project and pressing *OK* connects the local project (the one loaded by the interface) to the VC project (the version controlled project just selected). This is indicated by an informative message, while the button *Click to Disconnect* changes its caption to *Click to Connect* and the application title bar changes to include the name of the VC project your local project is connected to. Lastly (but not least), this will enable most other VC buttons (depending on your access rights).

Please note that every time you close EUROMOD, your local project will be automatically disconnected from the VC project.

Removing the connection

Once you have finished with any VC-related work you had to do, you can safely disconnect your local project by pressing the button *Click to Disconnect*. This will immediately disconnect your local project from the VC project it was connected to. The success of this action is indicated by an informative message, while the button *Click to Connect* changes its caption to *Click to Disconnect* and the application title bar changes to remove the name of the VC project your local project was connected to. Lastly (but not least), this will disable most other VC buttons.

[1] Note that you will not see at all the projects for which you have no access rights. For more information on VC access rights see *Administrate Users*

Download Bundle

This form allows users to synchronise their local project with content from a given VC project/bundle. You can access this form by clicking the *Download Bundle* button in the *Version Control* ribbon's section *PROJECT*.

The process of synchronise your local project is completed in three steps: choose the VC project, choose the bundle, choose the actions.

Choose the project

The *Project* combo-box on the top of the form, will allow you to select the VC project with which you want to synchronise your local project. If your local project is already connected to a VC project, then this VC project will appear preselected in the *Project* combo-box. Selecting a VC project will fill the *Bundle* combo-box with the available bundles for this project.

Choose the bundle

The *Bundle* combo-box (located just below the *Project* combo-box), will allow you to select a particular bundle (or version) of the VC project that you want to synchronise your local project with. You can get more info for a specific bundle after you select it by clicking the *Full Info* button next to it. Once you select a given bundle, the application will download all the bundle details (this may take a while) and compare each available unit to your local project. This information will then be displayed in the grid below, listing for each available unit its *Name*, *Type*, *Version* info (date & bundle it was commited to), its *Local Status* and a set of available actions that you can perform. The *Local Status* can be:

- *Corresponds*: meaning that the local unit is identical to the VC unit.
- *Differs*: meaning that the local unit is different to the VC unit.
- *Missing*: meaning that the local unit does not exist at all.

Choose the action for each unit

There are three possible actions for each unit:

- *Get Online Version*: meaning that you will download the VC unit and overwrite your local unit. Note that any not uploaded changes of the former local unit will be lost.

- *Keep Local Version*: meaning in fact no action, i.e. the VC unit is not downloaded.
- *Get Merge Support*: will download the VC unit in a special folder and build a merging environment allowing you to use the *Merge Tool* to consolidate the two versions (VC & local). For more information on merging see The Merge Tool.

Depending on the *Type* and *Local Status* of each unit, some of these actions may be disabled. For example if your local unit already *corresponds* to the VC unit, then all actions will be disabled as there is nothing further to be done. Similarly, if the unit type is not supported by the *Merge Tool*, the *Get Merge Support* action will be disabled.

In order to make the action selection process more efficient, just under the grid on the right side you will also find three buttons that allow you to change the selected action for all units with one click: *Get All*, *Keep All* and *Merge All*. These will of course only affect units for which the action is available.

Once you have selected all the required actions, you can click on the *Download* button to complete the synchronisation process.

Start/Finish Merging

If you are working on a given project and you have it on Version Control (VC), you will regularly find the need to save your work by uploading it to VC. You can do this by creating and uploading a new *Bundle*. This section describes how to merge (upload) a new bundle in a VC project. Note that in order to do the following, you need to already be logged in and connected to a VC project, and you need to have the appropriate access rights to upload a new merge for this project.

Start Merging

Pressing the button *Start Merging* (which is in the *PROJECT* group of the VC ribbon), will commence a new *Merge*. Note that only one active merge is allowed at any given time for a given VC project. If there is already someone else merging when you click the button, you will get an error message telling you which user is currently merging. If none else is merging, you will need to confirm that you want to start a new merge, and if you reply "Yes" a new merge will be started. This means that the VC project will be locked allowing only you to change it and an email will be sent (hopefully) to all the members of this project.

Finish Merging

Once the merge is started, you can make any final changes to you project (we recommend also downloading the latest available version in the VC project if you have not already done this, to make sure that you will not overwrite anyone else's changes) and then it is time to *Finish* the merge. Clicking the *Finish Merging* button will assess your local units and bring up the *Version Control - Upload Bundle* form.

This form has a list of available units locally & online, showing their status and allowing you to select which ones to upload. The status column can have one of three possible values:

- *Up-to-date*: This means that the local unit already matches the latest version in the VC project, so there is no further action that can be taken.
- *Differs*: This means that the local unit is different to the latest version in the VC project.
- *Pending*: This is a very special case, and it means that the local unit matches the latest online version of the unit, but this is not included in any bundles of the VC project. This can happen for example if one deleted a bundle (e.g. to correct the version number).

The Version Control will automatically produce a new version number for your bundle, based on the latest available version online. If you wish to change the version number, you can do so by checking the *Manual Version* checkbox on the middle-right and manually typing the version number your bundle should have.

Clicking the *Upload* button will upload all the selected units and generate the new bundle on the VC project. If everything is completed successfully it will also send an email to all the members of this VC project notifying them that your merge was completed and it will unlock the VC project allowing other users to merge in turn.

Abort Merging

If at any point you need to abort your current merge without changing anything in the VC project, you can do this by clicking on the *Abort Merging* button. This will immediately unlock the VC project and send an email to all the members of this VC project notifying them that your merge was aborted.

The Merge Tool

The user interface provides a tool that supports the merging of two versions of a country or add-on. Imagine developer A and developer B both have worked on country X. The tool firstly allows viewing what was changed between the two versions. Secondly, the tool offers facilities to determine which changes are to be overtaken in the final merged version.

Please note that the following description, for simplicity reasons, refers to countries only though all descriptions apply for add-ons as well.

Selecting the versions

Clicking the button *Merge Country* in the VC ribbon opens a dialog which allows defining the versions which are to be merged, whereupon one of the versions is always the country, which is currently loaded by the user interface and from which the dialog was started. This version is referred to as the ***Local Version***.

Consequently, the dialog allows for loading a ***Remote Version***, which is the country folder containing the version (i.e. XML-files) that should be merged with the *Local Version*.

Moreover, a ***Parent Version*** can be selected. This is the version from which as well the local as the remote version started from, i.e. ideally the base where the two versions diverged. This parent version is used by the interface (solely) to decide whether a change occurred in the local version or in the remote version. [1]

Taking into account, that a parent version may not always be available, the dialog allows that either the local (by checking the box *Use Local*) or the remote version (by checking the box *Use Remote*) may be defined as the parent version, meaning that all changes are considered as remote respectively local.

Clicking *OK* performs the comparison, which, depending on the extent of differences, may take a while, and opens the *merge dialog*.

The merge dialog

The merge dialog is split up into three areas:

- The (small) upper part displays changes in the country settings (see

Representation of changes in country settings)

- The (big) middle part displays all other changes (see *Representation of changes*)
- The (small) lower part comprises three buttons, which allow for performing the merge (see *Performing the merge - button Apply*) or interrupting one's work by optionally saving the so far taken definitions (see *Interrupting one's work - buttons Save and Close*).

Representation of changes

The middle and main part of the merge dialog comprises a register with the following four tabs:

- *Systems*: This tab lists added and deleted systems as well as changes in system settings.
- *Policy*-Spine: Is the main tab and displays all changes in the policy spine, i.e. added and deleted policies, functions and parameters; changes in policy/function-switches and parameter values; other changes in policies, functions and parameters, like e.g. changed name, comment, etc.
- *Data*: This tab lists added and deleted datasets as well as changes in dataset settings. In addition, it shows changes in the systems which can be used with the datasets. Moreover, it lists changes in the settings of policy-switches.
- *Conditional Formatting*: This tab lists changes in conditional formatting settings.

The structure of the four tabs is very similar and will be described by means of the main tab, the *Policy-Spine* tab.

The main part of each tab is taken by two "lists" (with tree-structure, to allow for collapsing and expanding the information), where the left one presents the local spine, whereas the right one presents the remote spine. The two lists are synchronised so that one can see local and remote spine side-by-side. That means, amongst others, that expanding or scrolling in one of the two lists always effects in the same move of the other list. Initially the two spines are an extract of the two spines, in the sense that only those parts of the spine are displayed which contain changes. Please note that, given the tree-structure of the list, this means that if e.g. a parameter is changed the parent-function and parent-policy is visible too. However, the sibling-parameters are not displayed, neither the "aunt"-functions (unless they show changes as well). This initial view can be further reduced by filtering changes, as described below, or extended to the full spine by pressing the button *Release Filter*.

The bottom part of the tab fulfils two tasks: The left-bottom-part allows for filtering the changes. That means it provides options to show only certain types of changes. As an example, one may want to see only added and removed policies, but not anything else (i.e. added/deleted functions/parameters should be not displayed, neither changes in values switches or other settings of policies, functions and parameters).

The right-bottom-part allows for accepting and rejecting changes, i.e. determining which changes are to be overtaken into the merged version. For more information see *Defining the carrying out of the merge* below.

Types and representation of changes

In principle, there are three types of changes:

- **Additions** (of policies, functions and parameters) are displayed by a green plus at the left edge of the spine.
- Accordingly, **removals** are displayed by a red minus.
- **Changes** are marked by a pencil. Note that changes may concern parameter-***values*** or policy/function-***switches***, as well as parameter/function/policy-***settings*** (like e.g. comments, private-settings, etc.). Accordingly, each policy, function and parameter is accompanied by columns containing these ***settings***, as well as the ***value/switch***-settings per system. This means that a "change" (display of a pencil) in e.g. a policy may mean that the policy switch is changed in one or more systems and/or that any policy-setting, like e.g. its name, is changed. Which of these settings is changed is represented by highlighting the respective cell with either red or green colour (the meaning of red and green will be explained in the paragraph after the next).

Whether a change is local or remote is represented by showing the plus/minus/pencil and the highlighting in either the (left) local tree or the (right) remote tree. Note that for added components the respective other tree shows an empty row (as the component does not exist in this version). For deleted components, the concerned tree shows the name (for better understanding), while the other tree (where the component still exists) shows the full information.

Alongside each change-symbol (plus/minus/pencil) there is another symbol, which indicates whether the change is accepted or rejected, i.e. should go into the merged version or not. Accepted changes are accompanied by a green hook, whereas rejected changes are accompanied by a red cross. Moreover, there is a third symbol, an arrow pointing to the right. This symbol may show up if e.g. a policy has several changes, i.e. the name and the comment is changed, and/or the switch is changed in more than one system. The arrow is displayed if these changes are partly accepted and partly rejected. Whether the changes are accepted or rejected, is represented by the colour of the highlighting. Green highlighting marks an accepted change, while red highlighting marks a rejected change.

Special display of order-changes

Given the fact that the local and the remote spine are synchronised, in the sense that "twin"-polices, functions and parameters are displayed side-by-side, the order of the spine is not necessarily correctly displayed, as it may differ between local and remote version.[2]

For watching the real order, and possibly assess differences in the order of the local and the remote spine, a special column can be shown. This is accomplished by clicking the button *Show Order*. The column shows entries like *LR:7* or *L:7/R:8*, where the former, if displayed with a function, means, that the function is in both spines the 7th within its parent-policy. The latter means, that the function is the 7th in the *L*ocal spine whereas it is the 8th in the *R*emote spine.

Note that "parallelism" is not affected by new or removed components. For example orders like local *PolA-PolB-PolC-PolD* and remote *PolA-PolB-**newPolBC**-PolC-PolD* would be numbered as *PolA(LR:1)-PolB(LR:2)-**newPolBC**(L:R:3)-PolC(LR:4)-PolD(LR:5).*

Also note that for a full view of the order you may want to show the full spine by clicking the button *Release Filter*.

As it would however be rather cumbersome to watch out for order changes in the order-column, policies and functions where the function- respectively parameter-order differs in local and remote spine are marked by a highlighted (green or red) button at the very left edge of the local spine. Differences in the order of policies are indicated by a highlighted button at the very left of the heading-row of the local spine.

Filtering changes

The merge tool offers, in the left-bottom part of the dialog, several options to hide and show different types of changes, with the intent to provide the user with a better overview.

Remote / Local:

- *Local and Remote*: With this option checked as well local as remote changes are displayed.
- *Local only*: With this option checked only local changes are displayed. For a better understanding note that the displays of the local and remote spine are still synchronised, i.e. show the same policies, functions and parameters. However, if for example a function is changed in the remote version, but not in the local version, this function will not be visible (neither in the local nor in the remote spine).
- *Remote only*: With this option checked only remote changes are displayed, in the same sense as described above for *Local only*.

Type:

- *Added*: Checking this option effects that added policies, functions and parameters are displayed (unless they are out-ruled by a *Level*- or *Remote/Local*-filter), while unchecking the option means that added policies are not displayed.
- *Removed*: Checking/unchecking this option effects the same for removed components as described above for added components.
- *Reordered*: Checking/unchecking this option effects the same for policies and function with different function- respectively parameter-order in the two spines as described above for added components.
- *Changed*: In principle checking/unchecking this option effects the same for

changed components as described above for added components. However, the filter can be further refined by showing only selected types of changes:

  - *All Settings / Selected Settings*: If the option *All Settings* is checked, all components with changes in any setting (name, comment, private, ...) are shown. This can however be restricted to, e.g. only show components where the comment is changed, by unchecking *All Settings* and therewith enable the check-boxes alongside the single settings. The task is accomplished by unchecking all single-settings except the *Comment*-setting (and perhaps the *Name*-setting for a better overview).
  - *All Values / Selected Values*: This option works in the same manner for parameter-values and policy/function-switches per system as described above for *Settings*.

Level:

- *Parameter*: Selecting this option corresponds in fact to no *Level*-filter.
- *Function*: Selecting this option effects that only changes on policy- and function level are displayed, i.e. parameter-level changes are hidden.
- *Policy*: Selecting this option effects that only changes on policy-level are displayed, i.e. function- and parameter-level changes are hidden.

Please note that the filters only take effect once the button *Apply / Update Filter* is clicked.

As mentioned above, the button *Release Filter* effects the display of the full spine, i.e. suspends all filters.

Defining the carrying out of the merge

Initially the merge tool suggests accepting all changes, i.e. overtake them into the merged version. This is reflected by green hooks and green highlighting. Only if there is a conflict, i.e. a component was changed in the local version as well as in the remote version, the local version is preferred, i.e. the remote version shows a red cross (or left-arrow) and possibly a red highlighting. This default setting can be changed in several ways.

Accepting / rejecting all changes

In the left-bottom part of the dialog the merge tool offers the buttons *Accept All* and *Reject All* to accept respectively reject all changes. For technical reasons (in context with conflict cases) accepting/rejecting all changes can be only done for one version at once. In other words, if one wants to accept *all*, i.e. local *and* remote, changes, one needs to firstly activate the option *Local*, then click the respective button, to repeat the same by activating the option *Remote*.

Still what "all" effectively means depends on the option *Visible Only*. If the option is not activated "all" actually means all. However, if the option is activated, filters are taken into account, which is described under *Effectiveness of filters* below.

Accepting all (local or remote) changes may lead to conflicts, where the tool needs the user's decision on what to do. For example, if all remote changes are to be accepted and a policy's comment was changed in the local version as well as in the remote version (i.e. both differ from the parent version), and the respective local change is currently set to being accepted, there is a conflict that cannot be solved by the tool without further information. For such cases, the handling is determined by the options under *Conflicts -Prefer ...*. If the option *Local* is selected, the tool will automatically keep the acceptance of the local change and not set the remote change to accept. If the option *Remote* is selected, the tool will set the remote change to accept and automatically reject the local change. If the option *Warn* is selected, the tool issues a warning for each conflict that it is about to act as if the option *Remote* (in case of *Accept All* remote changes) was selected and allows the user to refuse this doing.

Effectiveness of filters

If the option *Visible Only* is activated, filters are taken into account. For example, assume that the option *Added* is activated only, while the check-boxes for *Removed*, *Reordered* and *Changed* are not ticked. In this case the list of changes would only show added components. Respectively clicking for example *Accept All* will only accept the addition of components, but not the removal, change or reordering. Note however, that no definition set before the clicking of the button will be reset, i.e. an already accepted removal would not be changed to rejected.

For clarification, it should also be mentioned that "visible" refers to filters and not to actual visibility. That means missing visibility due to collapsing has no effect.

Accepting / rejecting single changes

Accepting and rejecting of changes can also be fine-tuned to single policies, functions and parameters and even to single settings, parameter-values and policy/function-switches. For this purpose, the merge tool provides context menus.

Right-clicking the change-symbols (plus/minus/pencil or hook/cross) left of the changed component opens a context menu with the options *Accept Changes*, *Reject Changes* and a check-box *Visible Only*. The logic behind this is the same as described under *Accepting / rejecting all changes*, just that the reference is the respective policy, function or parameter (i.e. all changed values/switches and settings) instead of *all*.

Right-clicking the cell of a single change, e.g. the change of the value of a parameter within a certain system, offers the same context menu, except that the option *Visible Only* is not available as it does not make sense. Again, the logic behind this is the same as described above, just that the reference in this case is the single change instead of all changes of the parameter's values and settings.

A very similar context menu is offered by right-clicking an "order change", i.e. one of the highlighted buttons at the very left edge of the local tree. In this case the context menu offers the options *Accept Order*, *Reject Order* and a check-box *Include Sub-Nodes*. Selecting *Accept Order* means that the local order is to be taken as the relevant for the merged version. Selecting *Reject Order* means that the local order is rejected and thus that the remote order is taken as the relevant for the merged version. Ticking the check-box *Include Sub-Nodes* effects that the relevant order for "children" (functions and parameters in the case of policies; parameters in the case of functions) is the same as that for the "parent". This means selecting an order for the button concerning the spine (the button alongside the column-headers) implies selecting an order for all policies, functions and parameters of the merged version.

Representation of changes in country settings

The upper part of the merge dialog displays the country-settings of the local (on the left side) and the remote version (right of it). Concretely it shows the name of the country (e.g. Austria, Hungary, Romania) and whether the country is "private" or not. If these settings are different in the local and the remote version, the changed-symbol (pencil) is displayed alongside the setting, together with either the accept-symbol (green hook) or the reject-symbol (red cross). Clicking this symbol opens a context menu with the options *Accept* and *Reject*.

Interrupting one's work - buttons Save and Close

If there are many differences between the local and the remote version it may take quite some consideration to decide what should go into the finally merged version. Therefore, the merge tool provides a *Save* button for saving all the so far taken decisions on accepting and rejecting changes. The merge tool and even the whole interface can then be closed without loss of the so far accomplished work. Upon the next time the merge tool is opened for the respective country, the user interface asks whether the "in-progress merging" should be used or not.

Note that opening the merge tool with a saved version also reduces the loading time as the comparison of the local and the remote version does not have to be repeated.

Also note that the question whether an "in-progress merging" is to be used is even asked if the merge tool is closed by the *Close* button, i.e. accepting/rejecting changes is not saved. The user interface then refers to the copies of the local-, remote- and parent-version of the country it made at the first opening of the merge tool (in a sub-folder of the country folder, called *Merge*). In this case the comparison of versions has to be repeated and loading is not accelerated.

As however these copies in the *Merge* folder are at risk to being outdated in the sense that the local version diverges from the actual country version, the interface performs a respective comparison and, if necessary, warns that "the info is possibly not up-to-date".

Performing the merge - button Apply

Pressing the *Apply* button will perform the merge by taking the decisions on accepting or rejecting changes into account. Once accomplished the user interface shows this merged version. All intermediate info (concretely the *Merge* folder within the country folder) is deleted.

Please note that the tool may show the info that "... the local XML-files used for merging are possibly not up-to-date". This happens if the country was changed while merging was in progress (see *Interrupting one's work - buttons Save and Close*). It is up to the user to decide whether she still wants to perform the merge.

Finally note that the changes caused by the merge cannot be undone by using the undo functionality. However, the user interface produces a backup before starting the action, which can be restored via the button *Restore* in the ribbon *Country Tools*. For more information see (Working with EUROMOD - Backup - Restore).

[1] Technically that means that, for a beginning, local and remote version are directly compared. A difference between the two is then considered as local change, if remote and parent version match and as remote change, if local and parent version match. If both, local and remote version, diverge from parent version, the change is classified as a "conflict".

[2] In fact the order may not reflect the real one for both spines, as for simplicity reasons it follows the location in the XML-file, which is arbitrary and especially for new components frequently different from the real order.

The Merge Tool for Variables

The user interface provides a tool that supports the merging of two versions of the EUROMOD variables file (see Working with EUROMOD - Administration of EUROMOD variables and EUROMOD Installation and Architecture - EUROMOD content (parameter files)). Imagine developer A and developer B both have edited EUROMOD variables. The tool firstly allows viewing what was changed by developer A and what by developer B.[1] Secondly, the tool offers facilities to determine which changes are to be overtaken in the final merged version.

Selecting the versions

Clicking the button *Merge Variables* (found in the *Administration Tools* or *Version Control* ribbon, depending on whether you have Version Control installed) opens a dialog which allows defining the versions which are to be merged, whereupon one of the versions is always the EUROMOD variables file, which is currently loaded by the user interface. This version is referred to as the ***Local Version***.

Consequently, the dialog allows for loading a ***Remote Version***, which is the file containing the EUROMOD variables (i.e. most likely another *VarConfig.xml* file) that should be merged with the *Local Version*.

Moreover, a ***Parent Version*** can be selected. This is the version from which as well the local as the remote version started from, i.e. ideally the base where the two versions diverged. This parent version is used by the interface (solely) to decide whether a change occurred in the local version or in the remote version.[2]

Taking into account, that a parent version may not always be available, the dialog allows that either the local (by checking the box *Use Local*) or the remote version (by checking the box *Use Remote*) may be defined as the parent version, meaning that all changes are considered as remote respectively local.

Clicking *OK* performs the comparison, which, depending on the extent of differences, may take a while. The duration of the check is considerably increased if all country-specific descriptions have to be compared. Therefore, a checkbox *Skip Country-Label Check* allows for skipping this check.

Once the comparison is finalised the *merge dialog* is opened.

The merge dialog

The merge dialog is split up into two areas.

- The (big) upper part displays the changes (see *Representation of changes*)
- The (small) lower part comprises three buttons, which allow for performing the merge (see *Performing the merge - button Apply*) or interrupting one's work by optionally saving the so far taken definitions (see *Interrupting one's work - buttons Save and Close*).

Representation of changes

The upper and main part of the merge dialog comprises a register with the following four tabs:

- *Variables*: This tab lists changes of the EUROMOD variables, i.e. added and deleted variables as well as changes in variables settings.
- *Acronyms*: This tab displays changes in the acronyms by which EUROMOD variables are built (see Working with EUROMOD - Administration of EUROMOD variables).
- *Country Labels*: This tab lists changes in the country-specific descriptions of the variables.
- *Switchable Policies*: This tab lists changes in EUROMOD's switchable policies (see Working with EUROMOD - Administrating policy switches).

The structure of the four tabs is very similar. The main part of each tab is taken by two "lists", where the left one presents the local version, whereas the right one presents the remote version. The two lists are synchronised so that one can see local and remote version side-by-side. That means, amongst others, that expanding or scrolling in one of the two lists always effects in the same move of the other list. Initially the two lists are an extract of the two versions, in the sense that only those parts are displayed which contain changes. This initial view can be further reduced by filtering changes, as described below.

The *Acronym*-tab is different from the other three tabs in the sense that it shows a tree-structure, i.e. acronyms (e.g. the acronym *ur* for urban) are arranged in acronym-levels (e.g. the acronym-level *region*), which are themselves arranged in acronym-types (e.g. the acronym-type *Demographic*).[3] This tree-structure allows for collapsing and expanding the information, and therewith for more clearness.[4]

Another mentionable, though seldom relevant, particularity concerns the *Country-Labels*-tab. If a whole country is added or removed from the user interface, this causes that (possibly empty) country-specific descriptions are added/removed to/from all variables. This is indicated by the entry "*Labels for country 'CC' added/removed*" in the list of changes.

The bottom part of the tab fulfils two tasks: The left-bottom-part allows for filtering the changes. That means it provides options to show only certain types of changes. As an example, one may want to see only added variables, but not anything else (i.e. deleted or changed variables).

The right-bottom-part allows for accepting and rejecting changes, i.e. determining which changes are to be overtaken into the merged version. For more information see *Defining the carrying out of the merge* below.

Types and representation of changes

In principle, there are three types of changes:

- **Additions** (of e.g. variables) are displayed by a green plus at the left edge of the list.
- Accordingly, **removals** are displayed by a red minus.
- **Changes** are marked by a pencil. A "change" of e.g. a variable may mean that the name of the variable is changed or that the monetary-status of the variable is changed. Which of these settings is changed is represented by highlighting the respective cell with either red or green colour (the meaning of red and green will be explained in the paragraph after the next).

Whether a change is local or remote is represented by showing the plus/minus/pencil and the highlighting in either the (left) local tree or the (right) remote tree. Note that for added components the respective other tree shows an empty row (as the component does not exist in this version). For deleted components the concerned tree shows the name (for better understanding), while the other tree (where the component still exists) shows the full information.

Alongside each change-symbol (plus/minus/pencil) there is another symbol, which indicates whether the change is accepted or rejected, i.e. should go into the merged version or not. Accepted changes are accompanied by a green hook, whereas rejected changes are accompanied by a red cross. Moreover, there is a third symbol, an arrow pointing to the right. This symbol may show up if e.g. a variable has several changes, i.e. the name and the monetary-status is changed. The arrow is displayed if these changes are partly accepted and partly rejected. Whether the changes are accepted or rejected, is represented by the colour of the highlighting. Green highlighting marks an accepted change, while red highlighting marks a rejected change.

Filtering changes

The merge tool offers, in the left-bottom part of the dialog, several options to hide and show different types of changes, with the intent to provide the user with a better overview.

Remote / Local:

- *Local and Remote*: With this option checked as well local as remote changes are displayed.
- *Local only*: With this option checked only local changes are displayed. For a better understanding note that the displays of the local and remote list are still synchronised, i.e. show the same components. However, if for example a variable is changed in the remote version, but not in the local version, this variable will not be visible (neither in the local nor in the remote list).
- *Remote only*: With this option checked only remote changes are displayed, in the same sense as described above for *Local only*.

Type:

- *Added*: Checking this option effects that added components are displayed (unless they are out-ruled by a *Level*- or *Remote/Local*-filter), while unchecking the option means that added components are not displayed.
- *Removed*: Checking/unchecking this option effects the same for removed components as described above for added components.
- *Changed*: In principle checking/unchecking this option effects the same for changed components as described above for added components. However,

the filter can be further refined by showing only selected types of changes:

  - *All Settings / Selected Settings*: If the option *All Settings* is checked, all components with changes in any setting are shown. This can however be restricted to, e.g. only show components where a certain setting (e.g. name) is changed, by unchecking *All Settings* and therewith enable the check-boxes alongside the single settings. The task is accomplished by unchecking all single-settings except the *Name*-setting.

Level: This option is only available in the *Acronyms*-tab.

- *Category*: Selecting this option corresponds in fact to no *Level*-filter.
- *Acronym*: Selecting this option effects that changes of acronym-categories are hidden.
- *Level*: Selecting this option effects that only changes on acronym-level are displayed, i.e. acronym- and category-level changes are hidden.
- *Type*: Selecting this option effects that only changes on acronym-type are displayed, i.e. acronym-level-, acronym- and category-level changes are hidden.

Please note that the filters only take effect once the button *Apply / Update Filter* is clicked.

The button *Release Filter* suspends all filters, i.e. all changes are displayed. Note that this still does not mean that e.g. all EUROMOD variables are displayed, but only variables which show any changes, were added or deleted. For technical reasons *Release Filter* pressed in the *Acronym*-tab effects, that all acronyms (including unchanged) are displayed.[5]

Defining the carrying out of the merge

Initially the merge tool suggests accepting all changes, i.e. overtake them into the merged version. This is reflected by green hooks and green highlighting. Only if there is a conflict, i.e. a component was changed in the local version as well as in the remote version, the local version is preferred, i.e. the remote version shows a red cross (or left-arrow) and possibly a red highlighting. This default setting can be changed in several ways.

Accepting / rejecting all changes

In the left-bottom part of the dialog the merge tool offers the buttons *Accept All* and *Reject All* to accept respectively reject all changes. For technical reasons (in context with conflict cases) accepting/rejecting all changes can be only done for one version at once. In other words, if one wants to accept *all*, i.e. local *and* remote, changes, one needs to firstly activate the option *Local*, then click the respective button, to repeat the same by activating the option *Remote*.

Still what "all" effectively means depends on the option *Visible Only*. If the option is not activated "all" actually means all. However, if the option is activated, filters are taken into account, which is described under *Effectiveness of filters* below.

Accepting all (local or remote) changes may lead to conflicts, where the tool needs the user's decision on what to do. For example, if all remote changes are to be accepted and a variable's name was changed in the local version as well as in the remote version (i.e. both differ from the parent version), and the respective local change is currently set to being accepted, there is a conflict that cannot be solved by the tool without further information. For such cases, the handling is determined by the options under *Conflicts -Prefer ...*. If the option *Local* is selected, the tool will automatically keep the acceptance of the local change and not set the remote change to accept. If the option *Remote* is selected, the tool will set the remote change to accept and automatically reject the local change. If the option *Warn* is selected, the tool issues a warning for each conflict that it is about to act as if the option *Remote* (in case of *Accept All* remote changes) was selected and allows the user to refuse this doing.

Effectiveness of filters

If the option *Visible Only* is activated, filters are taken into account. For example, assume that the option *Added* is activated only, while the check-boxes for *Removed* and *Changed* are not ticked. In this case the list of changes would only show added components. Respectively clicking for example *Accept All* will only accept the addition of components, but not the removal or change. Note however, that no definition set before the clicking of the button will be reset, i.e. an already accepted removal would not be changed to rejected.

For clarification, it should also be mentioned that "visible" refers to filters and not to actual visibility. That means missing visibility due to collapsing has no effect. This is however only relevant for the *Acronyms*-tab.

Accepting / rejecting single changes

Accepting and rejecting of changes can also be fine-tuned to single components and even to single settings. For this purpose, the merge tool provides context menus.

Right-clicking the change-symbols (plus/minus/pencil or hook/cross) left of the changed component opens a context menu with the options *Accept Changes*, *Reject Changes* and a check-box *Visible Only*. The logic behind this is the same as described under *Accepting / rejecting all changes*, just that the reference is the respective component (e.g. variable) instead of *all*.

Right-clicking the cell of a single change, e.g. the change of the name of a variable, offers the same context menu, except that the option *Visible Only* is not available as it does not make sense. Again, the logic behind this is the same as described above, just that the reference in this case is the single change instead of all changes of the variable's settings.

Interrupting one's work - buttons Save and Close

If there are many differences between the local and the remote version it may take quite some consideration to decide what should go into the finally merged version. Therefore, the merge tool provides a *Save* button for saving all the so far taken decisions on accepting and rejecting changes. The merge tool and even the whole interface can then be closed without loss of the so far accomplished work. Upon the next time the merge tool is opened, the user interface asks whether the "in-progress merging" should be used or not.

Note that opening the merge tool with a saved version also reduces the loading time as the comparison of the local and the remote version does not have to be repeated.

Also note that the question whether an "in-progress merging" is to be used is even asked if the merge tool is closed by the *Close* button, i.e. accepting/rejecting changes is not saved. The user interface then refers to the copies of the local-, remote- and parent-version it made at the first opening of the merge tool (in a sub-folder of the *Config*-folder, called *Merge*). In this case the comparison of versions has to be repeated and loading is not accelerated.

As however these copies in the *Merge* folder are at risk to being outdated in the sense that the local version diverges from the interface's actual version, the interface performs a respective comparison and, if necessary, warns that "the info is possibly not up-to-date".

Performing the merge - button Apply

Pressing the *Apply* button will perform the merge by taking the decisions on accepting or rejecting changes into account. Once accomplished the user interface shows this merged version. All intermediate info (concretely the *Merge* folder) is deleted.

Please note that the tool may show the info that "... the local XML-files used for merging are possibly not up-to-date". This happens if the variables file was changed while merging was in progress (see *Interrupting one's work - buttons Save and Close*). It is up to the user to decide whether she still wants to perform the merge.

Finally note that the changes caused by the merge cannot be undone by using the undo functionality. However, the user interface produces a backup before starting the action, which can be restored. For more information see (Working with EUROMOD - Backup - Restore).

[1] More precisely, the assignment of changes to one developer can only be accomplished if there is a common base version. Otherwise the user interface has to assume all changes to be done by one developer.

[2] Technically that means that, for a beginning, local and remote version are directly compared. A difference between the two is then considered as local change, if remote and parent version match and as remote change, if local and parent version match. If both, local and remote version, diverge from parent version, the change is classified as a "conflict".

[3] To be precise, there are four levels, as the acronym-level may be further subdivided into categories, e.g. the acronym *in* for industry into categories *agriculture*, *industry* and *services*.

[4] Please note that this means that if an acronym is changed the parent-level and parent-type is visible too. However, the sibling-acronyms are not displayed, neither the "aunt"-levels (unless they show changes as well).

[5] The "technical reason" is that the variables merge tool uses the same platform as the merge tool for countries/add-ons. In the context of a country it sometimes makes sense to see the whole spine, instead of only changed parts of the spine. In contrast, viewing all variables does not help to understand the change in a particular variable. Thus, to enhance performance, only the "relevant" (i.e. changed) components are loaded. For acronyms, there is no gain in performance in only loading the changed parts (rather, due to the tree-structure, it is more complicated), therefore they are completely loaded.

Administrate Content

Pressing the button *Administrate Content* in the *Version Control* ribbon's section *ADMIN* opens a dialog which displays all the units that are common between the VC project and the local project. This dialog allows for three operations:

a. Adding Local Units to the VC project

b. Downloading units of the VC project locally

c. Removing units from the VC project

Add Local Units to VC

Pressing this button lists units which ***exist locally***, but ***do not exist in the VC project***. The essential purpose of this operation is making a unit (existing in the loaded project) version controlled. By selecting the respective units from the list and pressing OK, the units will be "made version controlled", which, expressed more precisely, comprises two operations: Firstly, the local version of the unit is uploaded to VC and thus forms the first (remote) version of the unit. Secondly, the local

and the (new) remote version are linked.

Get Units from VC

Pressing this button lists units which ***do not exist locally***, however ***they exist in the VC project***. The essential purpose of this operation is to download a unit from the remote VC project and include it in your local project. By selecting the respective units from the list and pressing OK, the most recent version of the units will be "downloaded" and added to your local project.

Remove Units from VC

Pressing this button lists units which ***exist both locally and on the VC project***.

The essential purpose of this operation is to remove units from the VC project.

By selecting the respective units from the list and pressing OK, the units are removed from the VC project. Afterwards the user is asked, whether she

also wants to remove the units physically (i.e. delete them from the local project).

Administrate Users

Pressing the button *Administrate Users* in the *Version Control* ribbon's section *ADMIN* opens a dialog which allows for:

- Adding and removing users from the currently connected VC project
- Defining users' rights for the currently connected VC project

Please note that these operations are only available to users which have administration rights for the project (see below).

The main part of the dialog lists all users which are part of this VC project and for each of them their respective access rights.

Add Users

Pressing this button opens a dialog showing all users which potentially can be added to the project's VC. That means they are registered with the VC System but not yet users of the particular project. Ticking the respective users and pressing the button *OK* adds the users to this VC project, which means that they are now shown in the main-dialog's list of users. Initially the new users have no rights, what means in fact they have not yet access to any VC operation. For information on how to grant them the required rights see *Defining users' rights* below.

Remove Users

Users are removed from the project's VC by selecting them[1] and pressing the button *Remove Users*. The dialog asks for confirmation of this operation to avoid unintended removal.

Note that the actual addition or removal of users from the project's VC actually only takes place once the main-dialog is closed with *OK*. If the dialog is closed with *Cancel* no action takes place and thus all changes can still be undone.

Defining users' rights

A user can (or cannot) have the right to perform three types of version control operations: downloading units from VC, uploading units to VC and administrative operations. They are displayed and manipulated via the list of users in the *Version Control - Administrate Project Users* dialog.

Administration Right

This checkbox defines whether the listed user may administrate the project (ticked) or not (not ticked). Administration tasks are adding and removing units to/from the project's VC and administrating users and their rights. Users which have no such rights are informed of this fact once they try to perform any administrative operation.

Default Right

This column takes the values *Upload*, *Download* or *None*. The right can be changed by selecting another value from the list.

- *None*: This means that the user has no right to perform any VC operation. This makes in fact only sense, if the user is granted specific rights via the button *Refine Rights* (see below).
- *Download*: This means that the user has the right to download units (e.g. countries). She is however not allowed to upload any own versions to VC.
- *Upload*: This means that the user has the right to download units as well as upload own versions.

Refine Rights

Pressing this button opens a dialog which lists all available units, i.e. the countries, add-ons, etc., which are version controlled. Each of the units is accompanied by three checkboxes, headed with *Upload*, *Download* or *None*, where one of these headings is replaced by *Default* to inform that this is the user's default right. Initially this *Default* checkbox is ticked for all units. If required this can then be changed to e.g. grant a user with download-right as a default, upload-right on a specific country.

Some clarifications on not necessarily obvious mechanisms may be helpful:

- If a user downloads a release and has download-right as a default, however for some units "none-right", she will only download those units she has download-right on. Similarly, if a user with upload-right as a default uploads a release, and has on some units only download- or even none-right, she is only able to upload the units she has upload-right on. Note, that the user may not even notice these facts, as the not fitting units are just not displayed in the respective dialogs.

- If the default right is changed for a user who has some specific rights on certain units, these specifications are not lost. For example, a user initially has download-right as a default and for two units upload-right. If the default right is then changed to none, she will still have upload-right for the two units.

[1] The selection mechanism follows the usual Windows standards. That means for selecting several users, click them while holding the *Ctrl*-key. Alternatively, click the first user, press the shift key and select the last user. Selected users are marked by blue background colour.

Remove Bundles

From time to time you may find yourself in a situation (e.g. due to a human error) where you need to remove an already committed bundle. Assuming that you are already logged in, connected to a project and that you have admin rights, you can do this by clicking the *Remove Bundle* button to bring up the *Version Control - Remove Bundles* form.

The *Version Control - Remove Bundles* form displays a list with detailed information on all available bundles of this VC project. This list includes the exact date and time each bundle was generated, the version name, possibly some extra information given at the time of the bundle creation and the username of the *Author* of this particular bundle. Each bundle also has a checkbox on the left, allowing to select them.

Once you have selected the bundle(s) you want to remove, you can do so by clicking the *Remove Bundle(s)* button on the right. Note that removing the bundle will not actually remove the committed units - they will just no longer be connected to a specific bundle, so unless you change your local files, they will appear as *Pending* the next time you start a merge.

Add VC Project

You can create a new VC project by clicking the *Add VC Project* button, which you can find in the *ADMIN* section of the VC ribbon. In order to do this, you of course need to be logged in to Version Control and also to have the required access rights. Please note that the dialog suggests the name of the local project[1] (i.e. the project loaded by the user interface), however only if a VC project with this name does not already exist. Also note, that if you are already connected to a VC project, you will be asked to disconnect first. Pressing the button *OK* generates the new project remotely (i.e. on the version control system). This is confirmed by a respective success message and you are also automatically connected to the new VC project.

As a next step you would then need to click the *Administrate Content* button to add your local units to the new VC project (see Administrate Content).

Please note that the new VC project, after creation, has only one user, i.e. the user that generated the project. The user has full admin-rights on the project and my want to add other users by clicking the *Administrate Users* button (see Administrate Users).

[1] The name of a not version controlled project is the name of the folder where the project content is stored in.

Remove Project

To remove a project from the version control system, press the *Remove VC Project* button in the *ADMIN* group of the VC ribbon. Note that for this to work, you need to be already logged into Version Control and you need to have the required rights to remove projects. Pressing the *Remove VC Project* button opens the *Version Control - Remove Project* dialog. This lists all projects currently available on the version control system.[1] By selecting a project and pressing *OK* the respective project is completely and irrevocably removed from the version control system.

Note that the operation does not affect the local project (the project loaded by the user interface), even if this local project is linked to the just removed remote project. The only effect will be that you will be automatically disconnected from the VC project you just removed. For other users that may have been connected to this project when you removed it, the effect will be that the user interface issues error messages if one tries to operate version control operations. In this case one should disconnect from the removed VC project (see Connecting & Disconnecting a project) to avoid getting further error messages.

[1] For obvious reasons it does not list the EUROMOD core project(s).

Connecting to a VC project

The following explanations describe how most users will initially come into contact with EUROMOD version control. That is by, firstly, logging-in to the EUROMOD version control system and, secondly, by establishing a connection between their *local project* and *a VC project* stored on version control.

Logging-in to the EUROMOD version control system

The first time you go to the *Version Control* ribbon, you will see (at the very left) a closed door with the text *Click to Log In* underneath. Clicking this button will open the login dialog. The dialog asks you for a *User Name* and *Password*. Both should be at your disposal; if not, then you have to get them from the EUROMOD team. Once you enter the required information and click *OK*, the connection to the EUROMOD version control system is established.[1] Note that every time you close EUROMOD you are automatically logged out and you will need to repeat this process.

Now you are connected to the EUROMOD version control system. The user interface visualises this by showing an open door (with the text *Click to Log Out*) instead of the closed door. Moreover, you are now able to perform several actions (depending on your access rights), such as downloading a bundle, editing your personal VC information, or even some of the admin functions, such as creating/removing a project in VC. Still, in order to unlock the full capabilities of Version Control, you need to connect your local project to a VC project. The necessary steps to establish such a connection are described in the next section.

Establishing connection to a VC project

The *Version Control* ribbon shows in the section *PROJECT* a broken-chain symbol with the text *Click to connect* underneath. Following this instruction opens a dialog which offers all *available Version-Controlled (VC) Projects* for selection. Which projects are available and thus displayed depend on your user rights[2]. To connect your local project to one of the available VC projects, you need to select it and press *OK*.

Once your local project is connected with a VC project, this will show on the application's title bar where the text "Connected to VC: <ProjectName>" will be added to the local project information. Furthermore, you will now be able to complete more VC actions, such as starting a new merge or administering the VC project. Note that as soon as you connect to a VC project, you will get a notification in case there is someone currently performing a merge, or there is an unfinished merge pending by yourself.

Synchronising local project with a VC project

There are two reasons for synchronising your local project with a VC project: either you need to take the latest available version online to update your local project, or you may want to upload your latest changes to the VC project (or a combination of the two). If you want to update your local project, then you need to use the *Download Bundle* button. This will present you with the *Version Control - Download Bundle* form, allowing you to perform a different option for each unit (country, addon etc.) with a choice between keeping your local version, overwriting it with the online (VC) version or merging the two if both have new changes. If on the other hand you want to update the VC project with changes you have done locally, then you need to use the *Start Merge* button. This will present you with the *Version Control - Upload Bundle* form, allowing you to upload/update selected units creating a new bundle.[3]

[1] If you want to have closer information on logging-in and -out consult Logging in and out.

[2] See Administrate Version Control Users for more information about user rights.

[3] If you need to update a unit which has new changes both locally and online, then you need to first download and merge a final version locally and then upload it online.