Resources ▾

What is *euromod*?

EUROMOD - a tax-benefit microsimulation model for the EU.

Release Notes

PDF Documentation

Euromod Connector Toolbox™ is built to facilitate and simplify the usage of the EUROMOD microsimulation model for research and analysis purposes.

EUROMOD is a tax-benefit microsimulation model for the European Union 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. It is a static microsimulation model that applies user-defined tax and benefit policy rules to harmonised microdata on individuals and households, and calculates the effects of these rules on household income.

For more information, please visit the EUROMOD official webpage and the EUROMOD Documentation webpage.

Examples

Run simulations of Euromod tax-benefit policy-systems

User Guide

Documentation about the Euromod Connector toolbox

API reference

Description of main public classes and methods

License

Resources ▾

collapse all

Examples

Load and navigate the model

We start with loading the euromod toolbox and creating a Model object from a EUROMOD model. See the installation guide on how to download and install the toolbox and its dependencies.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
mod
```

```
1×1 Model with properties:

  extensions: [11×1 Extension]
   countries: [28×1 Country]
   modelpath: "C:\EUROMOD_RELEASES_I6.0+"
```

Note that every object that is related to the EUROMOD project comes with an informative description. Here we can see that the model has 3 relevant attributes to the user:

- countries
- extensions
- modelpath

The countries and extensions attributes contain elements of the respective objects. If we take a look at `countries`:

```
mod.countries
```

```
28×1 Country array:

      1: AT
      2: BE
      3: BG
      4: CY
      5: CZ
      6: DE
      7: DK
      8: EE
      9: EL
     10: ES
     11: FI
     12: FR
     13: HR
     14: HU
     15: IE
     16: IT
     17: LT
     18: LU
     19: LV
     20: MT
     21: NL
     22: PL
     23: PT
     24: RO
     25: se
     26: SI
     27: SK
     28: SL
```

We see indeed that the euromod model contains 28 countries. In a similar fashion we can look what kind of extensions are stored in the model. The countries class array can be indexed by both the number of the element and the country shortcode. Let us take a look at Sweden:

```
mod.countries("SE")
```

```
1×1 Country with properties:

        datasets: [27×1 Dataset]
      extensions: [12×1 Extension]
local_extensions: COVID
            name: "SE"
          parent: [1×1 Model]
        policies: [26×1 Policy]
         systems: [18×1 System]
```

Here we see again an informative representation of the Country class, which contains several properties that can be accessed. We can for example take a look at the first 10 policies that are stored in the country:

```
mod.countries("SE").policies(1:10)
```

```
10×1 Policy array:

   1: setdefault_se |                          | DEF: SET DEFAULT
   2: uprate_se     |                          | DEF: UPRATING FACTORS
   3: ConstDef_se   |                          | DEF: CONSTANTS
   4: IlsDef_se     |                          | DEF: INCOME CONCEPTS (standardized)
   5: IlsUDBdef_se  |                          | DEF: INCOME CONCEPTS (UDB)
   6: ildef_se      |                          | DEF: INCOME CONCEPTS (non-standardized)
   7: random_se     |                          | DEF: Random assignment
   8: TransLMA_se   |                          | DEF: Modelling labour market transitions (DO NOT SWITCH ON; ONLY WORKS WITH THE LMA ADD-ON)
   9: tudef_se      |                          | DEF: ASSESSMENT UNITS
  10: yem_se        | (with switch set for MWA) | DEF: minimum wage
```

Run with default configurations

Say that we are interested in running the tax system for the year 2021 of Sweden. Building further on the previous example we can look at the tax-systems contained in the model for Sweden:

Open Live Script

```
mod.SE.systems
```

```
18×1 System array:

    1: SE_2006
    2: SE_2007
    3: SE_2008
    4: SE_2009
    5: SE_2010
    6: SE_2011
    7: SE_2012
    8: SE_2013
    9: SE_2014
   10: SE_2015
   11: SE_2016
   12: SE_2017
   13: SE_2018
   14: SE_2019
   15: SE_2020
   16: SE_2021
   17: SE_2022
   18: SE_2023
```

In order to run the tax system we need a dataset that fits the requirement to use. The model however provides us with a list of datasets that are configured already:

```
mod.SE.SE_2021.datasets
```

```
5×1 DatasetInSystem array:

   1: training_data |
   2: SE_2019_a1    |
   3: SE_2020_b1    |
   4: SE_2021_hhot  |
   5: SE_2021_b1    | best match
```

Here we see that there are multiple datasets configured, but that `PL_2021_b1` is set as the best match dataset for this taxsystem. Provided that you have the microdata stored somewhere, you can then load it as a `table` and run the model in the following way:

```
data = readtable("SE_2021_b1.txt");
S=mod.("SE").("SE_2021").run(data,"SE_2021_b1");
S
```

```
1×1 Simulation with properties:

          outputs: {[21671×240 table]}
         settings: [1×1 struct]
 output_filenames: "se_2021_std"
           errors: [0×1 string]

output 1 : [21671x240 table]
   idhh   idperson   idmother   idfather   idpartner   idorighh   idorigperson   dag   dgn   dec
    200     20001          0          0           0        200          20001      45     1     0
    300     30001          0          0       30002        300          30001      26     1     0
    300     30002          0          0       30001        300          30002      26     0     0
    500     50001          0          0       50002        500          50001      37     1     0
    500     50002          0          0       50001        500          50002      33     0     0
    500     50003      50002      50001           0        500          50003       4     1     1
   ...
```

This returns a Simulation object with multiple properties. The one of interest here is outputs, which contains the output dataset(s) of type `table` returned by the microsimulation model.

```
outputdata_baseline = S.outputs{1};
outputdata_baseline(1:10,1:10)
```

```
10×10 table array:

   idhh    idperson    idmother    idfather    idpartner    idorighh    idorigperson    dag    dgn    dec
   ____    ________    ________    ________    _________    ________    ____________    ___    ___    ___

      200      20001           0           0            0         200           20001     45      1      0
      300      30001           0           0        30002         300           30001     26      1      0
      300      30002           0           0        30001         300           30002     26      0      0
      500      50001           0           0        50002         500           50001     37      1      0
      500      50002           0           0        50001         500           50002     33      0      0
      500      50003       50002       50001            0         500           50003      4      1      1
      600      60001           0           0            0         600           60001     44      1      0
      700      70001           0           0            0         700           70001     28      0      6
      900      90001           0           0            0         900           90001     77      1      0
     1300    1.3e+05           0           0      1.3e+05        1300         1.3e+05     54      0      0
```

Run with changing constants

One of the advantages of using the Matlab Connectors is the ability to run many counterfactual scenario's for the EUROMOD model. One can for example change the Tax Free income limit in Poland. There are multiple ways to do this via the euromod toolbox in Matlab, but one very straightforward way is to use the constantsToOverwrite Name-Value input argument:

Open Live Script

```
S=mod.('SE').('SE_2021').run(data,"SE_2021_b1",constantsToOverwrite={["$tinna_rate2",""],'0.4'});
```

```
Simulation SE_2021_std, system SE_2021, data SE_2021_b1, country SE: finished.
```

```
outputdata_changed= S.outputs{1};
sum(outputdata_changed.ils_dispy - outputdata_baseline.ils_dispy)
```

```
-1.2248e+07
```

The Name-Value input argument `constantsToOverwrite` specifies which constants to overwrite in the policy spline. It is a `N×1 cell` array of strings. The first element is itself a `1×2 string` indicating the targetted constant and the `order` (leave an empty string if it is `NaN`), and the second element is the value of type `string` to overwrite with. The default is [].

Run with add-ons

Run the simulation for the Swedish system `SE_2021` including the Marginal Tax-Rate add-on, using optional Name-Value input argument addons:

Open Live Script

```
S=mod.('SE').('SE_2021').run(data,"SE_2021_b1",addons=["MTR","MTR"]);
```

```
Simulation SE_2021_base_mtr, system se_2021_mtr, data SE_2021, country SE_2021_b1: finished.
```

```
S
```

```
1×1 Simulation with properties:

         outputs: {[21671×246 table]  [21671×39 table]}
        settings: [1×1 struct]
output_filenames: ["se_2021_base_mtr"    "se_2021_mtr"]
          errors: [0×1 string]

output 1 : [21671x246 table]
      idhh   idperson   idmother   idfather   idpartner   idorighh   idorigperson   dag   dgn   dec
       200     20001          0          0           0        200          20001      45     1     0
       300     30001          0          0       30002        300          30001      26     1     0
       300     30002          0          0       30001        300          30002      26     0     0
       500     50001          0          0       50002        500          50001      37     1     0
       500     50002          0          0       50001        500          50002      33     0     0
       500     50003      50002      50001           0        500          50003       4     1     1
      ...
output 2 : [21671x39 table]
      idhh   idperson        dwt yem_backup ils_earns_backup ils_earns_mtr_base1 ils_dispy_mtr_base1 ils_pen_mtr_base1 ils_benmt_mtr_base1 ils_be
       200     20001 9.200411e+02 5.634323e+04 5.634323e+04 5.634323e+04  6.499764e+04        0    0 7.019343e+03
       300     30001 8.892741e+02 4.004048e+04 4.089143e+04 4.089143e+04  3.055700e+04        0    0     0
       300     30002 8.892741e+02 2.400465e+04 2.448169e+04 2.448169e+04  1.974766e+04        0    0     0
       500     50001 9.082533e+02 4.805374e+04 4.805374e+04 4.805374e+04  3.600150e+04        0    0 1.675232e+03
       500     50002 9.082533e+02        0           0          0  2.794360e+03        0    0 2.794360e+03
       500     50003 9.082533e+02        0           0          0             0        0    0     0
      ...
```

As one can see there are two datasets returned by the model. Both of them can be accessed. The average marginal tax rate for example can be straightforwardly computed as:

```
mean(S.outputs{2}{:,'mtrpc'});
```

```
19.4191
```

The optional parameter `addons` that we passed to the `run` command is a `string` array of Name-Value arguments, with EUROMOD Addons to be integrated in the spine. For each Addon, specify the name of the Addon and the name of the system in the Addon to be integrated (typically, it is the *name of Addon_two-letter country code* e.g. LMA_AT). Default is [].

Run with extensions

Run the simulation for the Swedish system `SE_2021` switching on the Benefit Take-up Adjustment extension 'BTA' using the optional Name-Value input argument switches:

Open Live Script

```
S_BTA=mod.('SE').('SE_2021').run(data,"SE_2021_b1",switches=["BTA",true]);
```

```
Simulation SE_2021_base_mtr, system se_2021_mtr, data SE_2021, country SE_2021_b1: finished.
```

```
S
```

```
1×1 Simulation with properties:

          outputs: {[21671×246 table]  [21671×39 table]}
         settings: [1×1 struct]
  output_filenames: ["se_2021_base_mtr"    "se_2021_mtr"]
           errors: [0×1 string]

  output 1 : [21671x246 table]
        idhh   idperson   idmother   idfather   idpartner   idorighh   idorigperson   dag   dgn   dec
         200     20001          0          0           0        200          20001       45     1     0
         300     30001          0          0       30002        300          30001       26     1     0
         300     30002          0          0       30001        300          30002       26     0     0
         500     50001          0          0       50002        500          50001       37     1     0
         500     50002          0          0       50001        500          50002       33     0     0
         500     50003      50002      50001           0        500          50003        4     1     1
        ...
  output 2 : [21671x39 table]
        idhh   idperson        dwt yem_backup ils_earns_backup ils_earns_mtr_base1 ils_dispy_mtr_base1 ils_pen_mtr_base1 ils_benmt_mtr_base1 ils_
         200     20001 9.200411e+02 5.634323e+04 5.634323e+04 5.634323e+04  6.499764e+04        0    0 7.019343e+03
         300     30001 8.892741e+02 4.004048e+04 4.089143e+04 4.089143e+04  3.055700e+04        0    0     0
         300     30002 8.892741e+02 2.400465e+04 2.448169e+04 2.448169e+04  1.974766e+04        0    0     0
         500     50001 9.082533e+02 4.805374e+04 4.805374e+04 4.805374e+04  3.600150e+04        0    0 1.675232e+03
         500     50002 9.082533e+02        0           0          0  2.794360e+03        0    0 2.794360e+03
         500     50003 9.082533e+02        0           0          0             0        0    0     0
        ...
```

The optional parameter `switches` is a `string` array of Name-Value arguments, which define the EUROMOD extensions to be switched `on` or `off` in the simulation. For each extension specify the short name of the Extension and a `logical` value. Default is [].

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `System` | `Policy` | `Dataset` | `Extension` | `info` | `run`

Introduced in R2023a

Resources ▾

collapse all

User Guide

Release Notes

PDF Documentation

The Euromod Conector is a Matlab toolbox providing tools for running simulations and interacting with the tax-benefit microsimulation model EUROMOD.

Installation

Requirements

In order to run the model, we require two components: 1) the model (coded policy rules), and 2) the input microdata with the variables that respect the EUROMOD naming conventions. For more information, please, read the sections "Model" and "Input microdata" on the Download Euromod web page.

Running and navigating the model

The euromod toolbox is object oriented and evolves around using the Model class that loads a representation of the EUROMOD model. Create an object of the Model class by passing the path to the EUROMOD project:

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
mod
```

```
1×1 Model with properties:

  extensions: [11×1 Extension]
   countries: [28×1 Country]
   modelpath: "C:\EUROMOD_RELEASES_I6.0+"
```

Note that every object that is related to the EUROMOD project comes with an informative description. Here we can see that the model has 3 relevant attributes to the user:

- countries
- extensions
- modelpath

The countries and extensions attributes contain elements of the respective objects. If we take a look at `countries`:

```
mod.countries
```

```
28×1 Country array:

       1: AT
       2: BE
       3: BG
       4: CY
       5: CZ
       6: DE
       7: DK
       8: EE
       9: EL
      10: ES
      11: FI
      12: FR
      13: HR
      14: HU
      15: IE
      16: IT
      17: LT
      18: LU
      19: LV
      20: MT
      21: NL
      22: PL
      23: PT
      24: RO
      25: se
      26: SI
      27: SK
      28: SL
```

The `countries` property is a class array storing the Country objects and nesting the country-specific tax-benefit policies and systems. We see indeed that the euromod model contains 28 countries.

Note that the properties in a class array can be accessed by indexing the property with the element position, by the property name or by the ID.

Let us take have a look at Sweden:

```
mod.countries("SE")
```

```
1×1 Country with properties:

        datasets: [27×1 Dataset]
      extensions: [12×1 Extension]
local_extensions: COVID
            name: "SE"
          parent: [1×1 Model]
        policies: [26×1 Policy]
         systems: [18×1 System]
```

Here we see again an informative representation of the `Country` object, which contains several properties that can be accessed. In order to simulate a system we run a specific `System` object. We can obtain the systems for Sweden as follows:

```
mod.SE.systems
```

```
18×1 System array:

      1: SE_2006
      2: SE_2007
      3: SE_2008
      4: SE_2009
      5: SE_2010
      6: SE_2011
      7: SE_2012
      8: SE_2013
      9: SE_2014
     10: SE_2015
     11: SE_2016
     12: SE_2017
     13: SE_2018
     14: SE_2019
     15: SE_2020
     16: SE_2021
     17: SE_2022
     18: SE_2023
```

Running a simulation

In order to run the tax system we need a dataset that fits the requirement to use.

See the section Other important euromod objects on what datasets are configured and how.

If you know already which dataset to use you can simply load the data and run the model as follows:

```
data = readtable("SE_2021_b1.txt");
out_baseline=mod.("SE").("SE_2021").run(data,"SE_2021_b1");
out_baseline
```

```
1×1 Simulation with properties:

           outputs: {[21671×240 table]}
          settings: [1×1 struct]
  output_filenames: "se_2021_std"
            errors: [0×1 string]

 output 1 : [21671x240 table]
    idhh   idperson   idmother   idfather   idpartner   idorighh   idorigperson   dag   dgn   dec
     200     20001          0          0           0        200          20001    45     1     0
     300     30001          0          0       30002        300          30001    26     1     0
     300     30002          0          0       30001        300          30002    26     0     0
     500     50001          0          0       50002        500          50001    37     1     0
     500     50002          0          0       50001        500          50002    33     0     0
     500     50003      50002      50001           0        500          50003     4     1     1
    ...
```

Note that the run function here takes the mandatory input argument `dataset_id`, which in our case is `SE_2021_b1`. This is necessary such that EUROMOD can apply the dataset specific logic with respect to setting default values and uprating. Run returns a Simulation class with multiple properties. The one of interest here is outputs, which contains the outputdataset(s) of type `table` returned by the microsimulation model:

```
out_baseline(1:10,1:10)
```

```
10×10 table array:

idhh    idperson    idmother    idfather    idpartner    idorighh    idorigperson    dag    dgn    dec
____    ________    ________    ________    _________    ________    ____________    ___    ___    ___

 200      20001          0           0            0        200          20001        45      1      0
 300      30001          0           0        30002        300          30001        26      1      0
 300      30002          0           0        30001        300          30002        26      0      0
 500      50001          0           0        50002        500          50001        37      1      0
 500      50002          0           0        50001        500          50002        33      0      0
 500      50003      50002       50001            0        500          50003         4      1      1
 600      60001          0           0            0        600          60001        44      1      0
 700      70001          0           0            0        700          70001        28      0      6
 900      90001          0           0            0        900          90001        77      1      0
1300     1.3e+05         0           0       1.3e+05       1300         1.3e+05       54      0      0
```

Navigating the model

The `Model` object actually contains a full representation of the model that can be accessed using it's properties. The implementation in Matlab mimicks the hierarchical structure of the EUROMOD User Interface. A full description of the available types can be found in the API reference.

The spine

The spine of EUROMOD is what represents the series of calculations with respect to taxes and benefits. The spine consists out of three hierarchically ordered elements:

- Policy
- Function
- Parameter

The connector mimmicks this hierarchical implementation through an object-oriented representation. The three hierarchical elements are defined on the Country level and implemented on the System level.

Let us take a look at the policies, which are a property of the `Country` object:

```
mod.countries("SE").policies
```

```
26×1 Policy array:
    1: setdefault_se    |                          | DEF: SET DEFAULT
    2: uprate_se        |                          | DEF: UPRATING FACTORS
    3: ConstDef_se      |                          | DEF: CONSTANTS
    4: IlsDef_se        |                          | DEF: INCOME CONCEPTS (standardized)
    5: IlsUDBdef_se     |                          | DEF: INCOME CONCEPTS (UDB)
    6: ildef_se         |                          | DEF: INCOME CONCEPTS (non-standardized)
    7: random_se        |                          | DEF: Random assignment
    8: TransLMA_se      |                          | DEF: Modelling labour market transitions (DO NOT SWITCH ON; ONLY WORKS WITH THE LMA ADD-ON
    9: tudef_se         |                          | DEF: ASSESSMENT UNITS
   10: yem_se           | (with switch set for MWA) | DEF: minimum wage
   11: neg_se           |                          | DEF: recode negative self-employment income to zero
   12: yemcomp_se       |                          | BEN: wage compensation scheme COVID-19  (ONLY WORKING WITH LMA ADD-ON)
   13: bunct_se         |                          | BEN: unemployment benefit (contributory)
   14: bfapl_se         | (with switch set for PBE) | BEN: Parental leave benefit
   15: bpa_se           | (with switch set for PBE) | BEN: Special leave days other parent (10 days)
   16: tscee_se         |                          | SIC: Employee Social Insurance contribution
   17: tscer_se         |                          | SIC: Employer Social Insurance contribution
   18: tscse_se         |                          | SIC: Self-employed Social Insurance contribution
   19: tin_se           |                          | TAX: Personal Income tax
   20: tinkt_se         |                          | TAX: Tax on Capital Income
   21: bch_se           |                          | BEN: Child benefit
   22: bho_se           |                          | BEN: Housing allowance
   23: bhope_se         |                          | BEN: Housing allowance for pensioners
   24: bsamt_se         |                          | BEN: Social Assistance (means-tested)
   25: output_std_se    |                          | DEF: STANDARD OUTPUT INDIVIDUAL LEVEL
   26: output_std_hh_se |                          | DEF: STANDARD OUTPUT HOUSEHOLD LEVEL
```

As one can see the `policies` property is a class array therefore its elements, which are of type Policy here, are accessible by indexing:

```
mod.countries('SE').policies(13)
```

```
1×1 Policy with properties:

   comment: "BEN: unemployment benefit (contributory)"
 functions: [16×1 Function]
   private: "no"
extensions: [0×1 Extension]
        ID: "01f1c7ff-3b6d-4191-bc71-bb86db5603d6"
      name: "bunct_se"
     order: "13"
    parent: [1×1 Country]
spineOrder: "13"
```

The implementation of a policy is accessible through the System class:

```
mod.countries("SE").systems("SE_2021").policies
```

```
26×1 PolicyInSystem array:
    1: setdefault_se     | on                           | DEF: SET DEFAULT
    2: uprate_se         | on                           | DEF: UPRATING FACTORS
    3: ConstDef_se       | on                           | DEF: CONSTANTS
    4: IlsDef_se         | on                           | DEF: INCOME CONCEPTS (standardized)
    5: IlsUDBdef_se      | on                           | DEF: INCOME CONCEPTS (UDB)
    6: ildef_se          | on                           | DEF: INCOME CONCEPTS (non-standardized)
    7: random_se         | on                           | DEF: Random assignment
    8: TransLMA_se       | off                          | DEF: Modelling labour market transitions (DO NOT SWITCH ON; ONLY WORKS WITH THE LMA ADD
    9: tudef_se          | on                           | DEF: ASSESSMENT UNITS
   10: yem_se            | off (with switch set for MWA) | DEF: minimum wage
   11: neg_se            | on                           | DEF: recode negative self-employment income to zero
   12: yemcomp_se        | on                           | BEN: wage compensation scheme COVID-19  (ONLY WORKING WITH LMA ADD-ON)
   13: bunct_se          | off                          | BEN: unemployment benefit (contributory)
   14: bfapl_se          | off (with switch set for PBE) | BEN: Parental leave benefit
   15: bpa_se            | off (with switch set for PBE) | BEN: Special leave days other parent (10 days)
   16: tscee_se          | on                           | SIC: Employee Social Insurance contribution
   17: tscer_se          | on                           | SIC: Employer Social Insurance contribution
   18: tscse_se          | on                           | SIC: Self-employed Social Insurance contribution
   19: tin_se            | on                           | TAX: Personal Income tax
   20: tinkt_se          | on                           | TAX: Tax on Capital Income
   21: bch_se            | on                           | BEN: Child benefit
   22: bho_se            | on                           | BEN: Housing allowance
   23: bhope_se          | on                           | BEN: Housing allowance for pensioners
   24: bsamt_se          | on                           | BEN: Social Assistance (means-tested)
   25: output_std_se     | on                           | DEF: STANDARD OUTPUT INDIVIDUAL LEVEL
   26: output_std_hh_se  | off                          | DEF: STANDARD OUTPUT HOUSEHOLD LEVEL
```

Here we see that some policies are turned `off` by default. Note that the behaviour of the policies can be controlled from the connector. We can for example switch the policy `bunct_se` to `on`. Let us first look at the policy:

```
mod.countries('SE').systems('SE_2021').policies(13)
```

```
1×1 PolicyInSystem with properties:

    polID: "01f1c7ff-3b6d-4191-bc71-bb86db5603d6"
    Switch: "off"
     sysID: "bde78132-3f44-4d2e-a1ea-4849f88c2776"
   comment: "BEN: unemployment benefit (contributory)"
 functions: [16×1 FunctionInSystem]
   private: "no"
extensions: [0×1 Extension]
        ID: "bde78132-3f44-4d2e-a1ea-4849f88c277601f1c7ff-3b6d-4191-bc71-bb86db5603d6"
      name: "bunct_se"
     order: "13"
    parent: [1×1 System]
spineOrder: "13"
```

We see here the property `Switch` that is part of the PolicyInSystem class. This property, and similarly the other properties of the object, is modifiable. The changes that you will make will be passed to the EUROMOD software when simulating.

Note that currently, the Matlab connector does not support the save option. Therefore, changes implemented during a Matlab session cannot be saved.

To apply permanent changes to the model, we recommend using the User Interface of EUROMOD.

Note that the Matlab connector is not checking what kind of modifications you make to the model. Changing values of attributes like ID are definitely not recommended.

Here we see that the average benefit, which is represented by *ils_ben*, is indeed larger by switching on the extension of the *bunct_se* policy:

```
mod.countries("SE").systems("SE_2021").policies(14).Switch="on"
S=mod.countries("SE").systems("SE_2021").run(data,"SE_2021_b1")
mean(out_baseline.outputs{1}{:,"ils_ben"}) - mean(S.outputs{1}{:,"ils_ben"})
```

```
0.0
```

As mentioned before, the connector mimicks the hierarchical structure of the UI. Hence, the definition of `functions` and `parameters` are defined on the `Country` level, and their actual implementation are also accessible via the `Tax System`. Note that also here, the values of a `Parameter` and the `Switch` of a `Function` can be manipulated through the Matlab Connector without saving the changes permanently.

Display the functions defined in the *bho_se* policy:

```
mod.countries("SE").policies(23).functions
```

```
19×1 Function array:
    1: DefVar   | Temporary variables for Housing Allowance for pensioners
    2: Elig     | Living with partner
    3: ArithOp  | Wealth to be included in the means
    4: Elig     | Living without partner
    5: ArithOp  | Wealth to be included in the means
    6: Allocate | Allocation of wealth to the partners
    7: Elig     | Elderly or disabled adult (i.e. head or partner)
    8: BenCalc  | "Reserved amount (""income disregard"")"
    9: BenCalc  | Change in definition of Income Means for for Housing Allowance for pensioners (preliminary) since 2014
   10: DefIl    | Income Means for Housing Allowance for pensioners
   11: BenCalc  | Income of children is not take into account in the means
   12: BenCalc  | Maximum housing allowance
   13: BenCalc  |
   14: BenCalc  | Maximum housing allowance
   15: BenCalc  | Maximum housing allowance
   16: Allocate | Allocation of income to the partners
   17: BenCalc  | Final housing allowance for pensioners
   18: Allocate | Sharing housing cost
   19: BenCalc  | Housing allowance for pensioners
```

Display the functions defined in the *bho_se* policy and implemented in system *SE_2021*:

```
mod.countries("SE").systems("SE_2021").policies(23).functions
```

```
19×1 FunctionInSystem array:
    1: DefVar   | on   | Temporary variables for Housing Allowance for pensioners
    2: Elig     | on   | Living with partner
    3: ArithOp  | on   | Wealth to be included in the means
    4: Elig     | on   | Living without partner
    5: ArithOp  | on   | Wealth to be included in the means
    6: Allocate | on   | Allocation of wealth to the partners
    7: Elig     | on   | Elderly or disabled adult (i.e. head or partner)
    8: BenCalc  | on   | "Reserved amount (""income disregard"")"
    9: BenCalc  | on   | Change in definition of Income Means for for Housing Allowance for pensioners (preliminary) since 2014
   10: DefIl    | on   | Income Means for Housing Allowance for pensioners
   11: BenCalc  | on   | Income of children is not take into account in the means
   12: BenCalc  | on   | Maximum housing allowance
   13: BenCalc  | off  |
   14: BenCalc  | off  | Maximum housing allowance
   15: BenCalc  | on   | Maximum housing allowance
   16: Allocate | on   | Allocation of income to the partners
   17: BenCalc  | on   | Final housing allowance for pensioners
   18: Allocate | on   | Sharing housing cost
   19: BenCalc  | on   | Housing allowance for pensioners
```

"Overview of the parameters defined in the *bho_se* policy:

```
mod.countries("SE").policies(23).functions(1).parameters
```

```
1×1 Parameter with properties:
   comment: ""
extensions: [0×1 Extension]
     funID: "7eb7c6ef-a589-4725-bcce-3071684b38bb"
     group: "1"
        ID: "cecb57a5-273b-4ab0-b5cc-70dea318f4eb"
      name: "i_means_bhope_prel"
     order: "1"
    parent: [1×1 Function]
spineOrder: "23.1.1"
```

"Overview of the parameters defined in the *bho_se* policy and implemented in system *SE_2021*:

```
mod.countries("SE").systems("SE_2021").policies(23).functions(1).parameters
```

```
1×1 ParameterInSystem  with properties:
      parID: "cecb57a5-273b-4ab0-b5cc-70dea318f4eb"
      value: "0"
      sysID: "bde78132-3f44-4d2e-a1ea-4849f88c2776"
    comment: ""
 extensions: [0×1 Extension]
      funID: "7eb7c6ef-a589-4725-bcce-3071684b38bb"
      group: "1"
         ID: "bde78132-3f44-4d2e-a1ea-4849f88c2776cecb57a5-273b-4ab0-b5cc-70dea318f4eb"
       name: "i_means_bhope_prel"
      order: "1"
     parent: [1×1 FunctionInSystem]
 spineOrder: "23.1.1"
```

Other Important EUROMOD Objects

Central to the EUROMOD project, next to the coding of the `policies` is the microdata. How datasets should be treated by the model is configured in the model already. The property `datasets` is a `class` array with properties accessible and modifiable just like the spine-elements.

```
mod.SE.datasets
```

```
27×1 Dataset array:

       1: SE_2007_a4
       2: SE_2008_a3
       3: training_data
       4: SE_2010_a1
       5: SE_2012_a2
       6: SE_2015_a1
       7: SE_2009_hhot
       8: SE_2010_hhot
       9: SE_2011_hhot
      10: SE_2012_hhot
      11: SE_2013_hhot
      12: SE_2014_hhot
      13: SE_2015_hhot
      14: SE_2016_hhot
      15: SE_2017_hhot
      16: SE_2016_a1
      17: SE_2018_hhot
      18: SE_2019_hhot
      19: SE_2018_a2
      20: SE_2017_a3
      21: SE_2020_hhot
      22: SE_2019_a1
      23: SE_2020_b1
      24: SE_2021_hhot
      25: SE_2022_hhot
      26: SE_2021_b1
      27: SE_2023_hhot
```

In the previous section we used *SE_2021_b1*. Let us have a look at it:

```
mod.SE.datasets("SE_2021_b1")
```

```
1×1 Dataset with properties:

   coicopVersion: ""
         comment: ""
        currency: "national"
     decimalSign: "."
              ID: "c7b651ed-b311-4e39-80b4-18ca19957ce7"
listStringOutVar: ""
            name: "SE_2021_b1"
          parent: [1×1 Country]
         private: "no"
  readXVariables: "no"
useCommonDefault: "no"
  yearCollection: "2021"
         yearInc: "2020"
```

Similarly to the properties in the Policy, Function and Parameter objects, the properties of the `Dataset` can be modified here.

We can further check what datasets are implemented for a given system, for example *SE_2021*, as follows:

```
mod.SE.SE_2021.datasets
```

```
5×1 DatasetInSystem array:

   1: training_data |
   2: SE_2019_a1    |
   3: SE_2020_b1    |
   4: SE_2021_hhot  |
   5: SE_2021_b1    | best match
```

Another important concept in EUROMOD are `extensions` that are defined globally on the Model level:

```
mod.extensions
```

```
11×1 Extension array:

   1: BTA      | Benefit Take-up Adjustments
   2: TCA      | Tax Compliance Adjustments
   3: FYA      | Full Year Adjustments
   4: UAA      | Uprating by Average Adjustment
   5: EPS      | Extended Policy Simulation
   6: PBE      | Parental leave benefits
   7: MWA      | Minimum Wage Adjustments
   8: HHoT_un  | HHoT unemployment extension
   9: WEB      | EUROMOD JRC-Interface
  10: HHoT_ext | HHoT - Extended Simulation
  11: HHoT_ncp | HHoT - Non Compulsory Payments
```

Or locally on the Country level:

```
mod.countries("SE").extensions
```

```
12×1 Extension array:

   1: COVID     | COVID benefit
   2: BTA       | Benefit Take-up Adjustments
   3: TCA       | Tax Compliance Adjustments
   4: FYA       | Full Year Adjustments
   5: UAA       | Uprating by Average Adjustment
   6: EPS       | Extended Policy Simulation
   7: PBE       | Parental leave benefits
   8: MWA       | Minimum Wage Adjustments
   9: HHoT_un   | HHoT unemployment extension
  10: WEB       | EUROMOD JRC-Interface
  11: HHoT_ext  | HHoT - Extended Simulation
  12: HHoT_ncp  | HHoT - Non Compulsory Payments
```

The `extensions` property is a `class` array. If we want to access the information stored in the *Minimum Wage Adjustments* extension for example, we can simply use the following command:

```
mod.countries("SE").extensions("Minimum Wage Adjustments")
```

```
1×1 Extension with properties:

       ID: "557c232a-9ce6-4808-b52f-ca5e02fe8cf4"
     name: "Minimum Wage Adjustments"
   parent: [1×1 Country]
shortName: "MWA"
```

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `System` | `Policy` | `Dataset` | `Extension` | `info` | `run`

Introduced in R2023a

Resources

Euromod classes and methods

This reference guide lists the main public objects of the Euromod Connector toolbox. It provides useful functionalities that allow the user to interact with EUROMOD and run simulations.

Please, refer to the User Guide and Examples for futher readings.

`Country`	`class`	Countries with EUROMOD tax-benefit systems.
`Dataset`	`class`	Datasets configured in a country.
`DatasetInSystem`	`class`	Datasets configured in a system.
`euromod`	`method`	Method instantiating the base class Model.
`Extension`	`class`	EUROMOD model extensions.
`ExtensionSwitch`	`class`	Extension switches of an object.
`Function`	`class`	Functions in a policy.
`FunctionInSystem`	`class`	Functions configured in a system.
`Model`	`class`	Base class returned by method euromod().
`Parameter`	`class`	Parameters in a function.
`ParameterInSystem`	`class`	Parameters configured in a system.
`Policy`	`class`	Policy rules in a country.
`PolicyInSystem`	`class`	Policy rules configured in a system.
`ReferencePolicy`	`class`	Reference policies in a country.
`run`	`method`	Method for running simulations of a EUROMOD tax-benefit system.
`Simulation`	`class`	Simulation results returned by run.
`System`	`class`	Tax-benefit systems in a country.

Resources ▾

Country

country-specific EUROMOD tax-benefit models

Syntax

```
C = Country(Model)
```

Description

`C = Country(Model)` return a class array with the EUROMOD default country models. example

This class is instantiated authomatically when loading the Model and is stored there under attribute countries.

Examples

collapse all

Load EUROMOD countries

`Country` loads all the countries configured in EUROMOD by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
C=Country(mod);
C
```

```
28×1 Country array:

        1: AT
        2: BE
        3: BG
        4: CY
        5: CZ
        6: DE
        7: DK
        8: EE
        9: EL
       10: ES
       11: FI
       12: FR
       13: HR
       14: HU
       15: IE
       16: IT
       17: LT
       18: LU
       19: LV
       20: MT
       21: NL
       22: PL
       23: PT
       24: RO
       25: SE
       26: SI
       27: SK
       28: SL
```

Display a specific country in the `Model`.

```
C("AT")
```

```
1×1 Country with properties:

          datasets: [27×1 Dataset]
        extensions: [13×1 Extension]
  local_extensions: [2×1 Extension]
              name: "AT"
            parent: [1×1 Model]
          policies: [54×1 Policy]
           systems: [17×1 System]
```

Input Arguments

collapse all

`Model` — Euromod base class `Model`.
`class`

The `Model` class returned by euromod.

Data Types: `class`

Properties

collapse all

`datasets` — Dataset class

`class array`

Returns a `Dataset` class array with country datasets.

Example: `C.datasets`

`extensions` — Extension class

`class array`

Returns an `Extension` class array with the `Model` and `Country` extensions.

Example: `C.extensions`

`local_extensions` — Extension class

`class array`

Returns an `Extension` class array with the `Country` extensions.

Example: `C.local_extensions`

`name` — Two-letter country code

`string scalar`

Two-letter country code. See the Eurostat Glossary:Country codes.

Example: `C.name`

`parent` — Model base class

`class`

Returns the `Model` base class.

Example: `C.parent`

`policies` — Policy class

`class array`

Returns a `Policy` class array with the `Country` policies.

Example: `C.policies`

systems — System class
`class array`

Returns a `System` class array with the `Country` systems.

Example: `C.systems`

Object Functions

collapse all

info

`info` returns basic information on `Country` properties of type `class`.

Example: `mod.countries('AT').info('systems')`

run

`run` returns simulation results as a `Simulation` class. It requires 3 input arguments: system name, dataset, and dataset name.

Example: `mod.countries('SE').run('SE_2021',data,data_id)`

Output Arguments

collapse all

C — Country class
class array

A class array with the EUROMOD default countries.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `System` | `Policy` | `Dataset` | `Extension` | `info` | `run`

Introduced in R2023a

Resources ▾

Dataset

Datasets available in a EUROMOD country model.

Syntax

```
D = Dataset(Country)
```

Description

`D = Dataset(Country)` create a class array with the `Country` default datasets. example

This class contains the datasets implemented in a Country model. It is stored as property datasets of the `country` class.

This class is the superclass of the DatasetInSystem class.

Examples

collapse all

Load Country datasets

`Dataset` loads all the datasets configured in a `Country` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
D=Dataset(mod.countries("AT"));
D
```

```
27×1 Dataset array:

        1: training_data
        2: AT_2008_a8
        3: AT_2010_a4
        4: AT_2012_a8
        5: AT_2014_a6
        6: AT_2014_hhot
        7: AT_2015_hhot
        8: AT_2009_hhot
        9: AT_2010_hhot
       10: AT_2011_hhot
       11: AT_2012_hhot
       12: AT_2013_hhot
       13: AT_2016_hhot
       14: AT_2017_hhot
       15: AT_2015_a3
       16: AT_2016_a3
       17: AT_2018_hhot
       18: AT_2017_a3
       19: AT_2019_hhot
       20: AT_2018_a1
       21: AT_2020_hhot
       22: AT_2019_b2
       23: AT_2021_hhot
       24: AT_2020_b2
       25: AT_2022_hhot
       26: AT_2021_b1
       27: AT_2023_hhot
```

Display a specific dataset in the `Country`.

```
D("AT_2021_b1")
```

```
1×1 Dataset with properties:

       coicopVersion: ""
             comment: ""
            currency: "euro"
         decimalSign: "."
                  ID: "b781f486-9a8a-4704-a9bd-51e5cccec985"
    listStringOutVar: "sid_h sft_h"
                name: "AT_2021_b1"
              parent: [1×1 Country]
             private: "no"
       readXVariables: "no"
    useCommonDefault: "no"
      yearCollection: "2021"
             yearInc: "2020"
```

Input Arguments

collapse all

`Country` — `Country` class

`class`

A specific country from the Country class array.

Data Types: `class`

Properties

collapse all

`coicopVersion` — COICOP version

`string scalar`

Returns the COICOP version.

Example: `D(end).coicopVersion`

`comment` — Dataset comment

`string scalar`

Returns the comment about the dataset.

Example: `D(end).comment`

`currency` — Currency of values

`string scalar`

Returns the currency of the monetary values in the dataset.

Example: `D(end).currency`

`decimalSign` — Decimal delimiter

`string scalar`

Returns the decimal delimiter of float values.

Example: `D(end).decimalSign`

`ID` — Identifier number

`string scalar`

Returns the dataset identifier number.

Example: `D(end).ID`

`listStringOutVar` — Names of variables

`string scalar`

Returns the names of variables.

Example: `D(end).listStringOutVar`

`name` — Name of the dataset
`string scalar`

Returns the name of the dataset.

Example: `D(end).name`

`parent` — Country class.
`class`

Returns the specific Country class.

Example: `D(end).parent`

`private` — Access type
`string scalar`

Returns the access type of the dataset.

Example: `D(end).private`

`readXVariables` — Read variables
`string scalar`

Returns the loaded variables.

Example: `D(end).readXVariables`

`useCommonDefault` — Use default
`string scalar`

Returns the use default.

Example: `D(end).useCommonDefault`

`yearCollection` — Year of Dataset
`string scalar`

Returns the year of the dataset collection.

Example: `D(end).yearCollection`

`yearInc` — Year of variables
`string scalar`

Returns the reference year for the income variables.

Example: `D(end).yearInc`

Output Arguments

collapse all

`D` — Dataset class
class array

A class array with the `Country` default datasets.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `DatasetInSystem` | `bestmatchDatasets`

Introduced in R2023a

Resources

DatasetInSystem

datasets available in a EUROMOD system

Syntax

```
D = DatasetInSystem(System)
```

Description

`D = DatasetInSystem(System)` create a class array with the `System` default datasets. example

This class contains the datasets modelled in a EUROMOD system. The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. `name`, `ID`, `bestMatch`, etc.).

This class is stored in the property datasets of the System class.

This class inherits methods and properties from the superclass Dataset.

Examples

collapse all

Load System datasets

`DatasetInSystem` loads all the datasets configured in a `System` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
D=DatasetInSystem(mod.countries("AT").systems(end));
D
```

```
4×1 DatasetInSystem array:

        1: training_data |
        2: AT_2020_b2    |
        3: AT_2021_b1    | best match
        4: AT_2023_hhot  |
```

Display a specific dataset in the `System`.

```
D("AT_2021_b1")
```

```
1×1 DatasetInSystem with properties:

       bestMatch: "yes"
          dataID: "b781f486-9a8a-4704-a9bd-51e5cccec985"
           sysID: "064cf5eb-73bd-4fae-a421-ed9781e418c9"
   coicopVersion: ""
         comment: ""
        currency: "euro"
     decimalSign: "."
              ID: "064cf5eb-73bd-4fae-a421-ed9781e418c9b781f486-9a8a-4704-a9bd-51e5cccec985"
listStringOutVar: ""
            name: "AT_2021_b1"
          parent: [1×1 System]
         private: "no"
  readXVariables: "no"
useCommonDefault: "no"
  yearCollection: "2021"
         yearInc: "2020"
```

Input Arguments

collapse all

`System` — `System` class

`class`

A specific system from the System class array.

Data Types: `class`

Properties

collapse all

`bestMatch` — Best match datasets

`string scalar`

If yes, the current dataset is a best match for the specific system.

Example: `D(end).bestMatch`

`coicopVersion` — COICOP version

`string scalar`

Returns the COICOP version.

Example: `D(end).coicopVersion`

`comment` — DatasetInSystem comment

`string scalar`

Returns the comment about the dataset.

Example: `D(end).comment`

`currency` — Currency of values
`string scalar`

Returns the currency of the monetary values in the dataset.

Example: `D(end).currency`

`dataID` — Identifier number
`string scalar`

Returns the identifier number of the dataset at the system level.

Example: `D(end).dataID`

`decimalSign` — Decimal delimiter
`string scalar`

Returns the decimal delimiter of float values.

Example: `D(end).decimalSign`

`ID` — Identifier number
`string scalar`

Returns the dataset identifier number.

Example: `D(end).ID`

`listStringOutVar` — Names of variables
`string scalar`

Returns the names of variables.

Example: `D(end).listStringOutVar`

`name` — Name of the dataset
`string scalar`

Returns the name of the dataset.

Example: `D(end).name`

`parent` — System class.
`class`

Returns the specific System class.

Example: `D(end).parent`

`private` — Access type
`string scalar`

Returns the access type of the dataset.

Example: `D(end).private`

`readXVariables` — Read variables
`string scalar`

Returns the loaded variables.

Example: `D(end).readXVariables`

`sysID` — Identifier number
`string scalar`

Returns the identifier number of the reference system.

Example: `D(end).sysID`

`useCommonDefault` — Use default
`string scalar`

Returns the use default.

Example: `D(end).useCommonDefault`

`yearCollection` — Year of DatasetInSystem
`string scalar`

Returns the year of the dataset collection.

Example: `D(end).yearCollection`

`yearInc` — Year of variables
`string scalar`

Returns the reference year for the income variables.

Example: `D(end).yearInc`

Output Arguments

collapse all

`D` — DatasetInSystem class
class array

A class array with the `System` default datasets.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `System` | `Dataset` | `bestmatchDatasets`

Introduced in R2023a

Resources ▾

euromod

Load the microsimulation model EUROMOD.

Syntax

```
mod = euromod(model_path)
```

Description

`mod = euromod(model_path)` instantiates a base class `Model` with the microsimulation model EUROMOD. example

Input Arguments

collapse all

`model_path` — Path to EUROMOD files
`character vector` | `string scalar`

Full path to the root directory of the EUROMOD model files.

Data Types: `char` | `string`

Output Arguments

collapse all

`mod` — Model class
class

Returns an instance of the base class Model.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`Country` | `System` | `Policy` | `Dataset` | `Function` | `Parameter`

Introduced in R2023a

Resources ▾

Extension

EUROMOD extensions

Syntax

```
E = Extension(Parent)
```

Description

example

`E = Extension(Parent)` create a class array with the EUROMOD default extensions. The extensions can be at the model level if `Parent` is the Model class or at the country level if `Parent` is the Country class.

The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. name, ID, or shortName).

Examples

collapse all

Load Country extensions

`Extension` loads all the extensions configured in a `Country` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
E=Extension(mod.countries("AT"));
E
```

```
13×1 Extension array:

        1: SORESI    | SORESI web-based model
        2: LMA       | Labour market adjustments
        3: BTA       | Benefit Take-up Adjustments
        4: TCA       | Tax Compliance Adjustments
        5: FYA       | Full Year Adjustments
        6: UAA       | Uprating by Average Adjustment
        7: EPS       | Extended Policy Simulation
        8: PBE       | Parental leave benefits
        9: MWA       | Minimum Wage Adjustments
       10: HHoT_un   | HHoT unemployment extension
       11: WEB       | EUROMOD JRC-Interface
       12: HHoT_ext  | HHoT - Extended Simulation
       13: HHoT_ncp  | HHoT - Non Compulsory Payments
```

Display a specific extension in the `Country`.

```
E("MWA")
```

```
1×1 Extension with properties:

          ID: "557c232a-9ce6-4808-b52f-ca5e02fe8cf4"
        name: "Minimum Wage Adjustments"
      parent: [1×1 Model]
   shortName: "MWA"
```

Load Model extensions

`Extension` loads all the extensions configured in the `Model` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
E=Extension(mod);
E
```

```
11×1 Extension array:

         1: BTA       | Benefit Take-up Adjustments
         2: TCA       | Tax Compliance Adjustments
         3: FYA       | Full Year Adjustments
         4: UAA       | Uprating by Average Adjustment
         5: EPS       | Extended Policy Simulation
         6: PBE       | Parental leave benefits
         7: MWA       | Minimum Wage Adjustments
         8: HHoT_un   | HHoT unemployment extension
         9: WEB       | EUROMOD JRC-Interface
        10: HHoT_ext  | HHoT - Extended Simulation
        11: HHoT_ncp  | HHoT - Non Compulsory Payments
```

Input Arguments

collapse all

`Parent` — Euromod base class `Model`.

`class`

Can be the `Model` class returned by euromod or a specific `Country` class.

Data Types: `class`

Properties

collapse all

`ID` — Identifier number

`string scalar`

Returns the identifier number of the extension.

Example: `E(end).ID`

`name` — Long name
`string scalar`

Returns the long name of the extension.

Example: `E(end).name`

`parent` — Model base class
`class`

Returns the `Model` base class of EUROMOD.

Example: `E(end).parent`

`shortName` — Short name
`string scalar`

Returns the short name of the extension.

Example: `E(end).shortName`

Output Arguments

collapse all

`E` — Extension class
class array

A class array with the EUROMOD default extensions at the `Model` or `Country` level.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `ExtensionSwitch` | `local_extensions`

Introduced in R2023a

Resources ▾

Function

Functions of a EUROMOD tax-benefit policy.

Syntax

```
F = Function(Policy)
```

Description

`F = Function(Policy)` create a class array with the functions modelled in a `Policy`. example

This class contains functions implemented in a specific tax- benefit policy. The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. `name`, `ID`, `order`, etc.).

This class is stored in the property functions of the Policy class.

This class stores classes of type Extension and Parameter.

This class is the superclass of the FunctionInSystem class.

Examples

collapse all

Load Policy functions

`Function` loads all the functions configured in a `Policy` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
F=Function(mod.countries("AT").policies(8));
F
```

```
27×1 Function array:

        1: DefConst |   | ESTAT data: unemployment, pre-COVID input data
        2: DefConst |   | ESTAT data: unemployment, post-COVID input data
        3: DefConst |   | ESTAT data: MC for employees & self-employed, pre-COVID input data
        4: DefConst |   | ESTAT data: MC for employees & self-employed, post-COVID input data
        5: Elig     |   | eligibility conditions - transition from non-employed to employed
        6: BenCalc  |   | random allocation (based on ESTAT data) - transition non-employed to employed
        7: Elig     |   | eligibility conditions - transition from employed to unemployed
        8: BenCalc  |   | random allocation (based on ESTAT data) - transition employed to unemployed for employees
        9: Elig     |   | eligibility conditions - transition from self-employed to unemployed
       10: BenCalc  |   | random allocation - transition from self-employed to unemployed
       11: Elig     |   | eligibility conditions - yem_a
       12: ArithOp  |   | define yem_a
       13: Elig     |   | eligibility conditions - lhw_a
       14: ArithOp  |   | define lhw_a
       15: BenCalc  |   | set yemmy_a
       16: Elig     |   | step 1: eligibility conditions: MC for self-employed
       17: BenCalc  |   | step 2: random allocation (based on Eurostat statistics)
       18: Elig     |   | selected in step 2 -> eligible for step 3
       19: BenCalc  |   | step 3a: random allocation of months in compensation scheme (based on external statistics and/or assumptions)
       20: ArithOp  |   | step 3b: months out of compensation scheme (based on step 3a)
       21: Elig     |   | MC_EE step 1: eligibility conditions: MC for employees
       22: BenCalc  |   | MC_EE  step 2: random allocation (based on ESTAT statistics)
       23: Elig     |   | MC_EE selected in step 2 -> eligible for step 3
       24: BenCalc  |   | MC_EE step 3a: random allocation of months in MC (based on ESTAT statistics)
       25: ArithOp  |   | MC_EE step 3b: months out of compensation scheme (based on step 3a)
       26: Elig     |   | MC_EE  selected in step 3 -> eligible for step 4
       27: BenCalc  |   | MC_EE  step 4: share of hours worked in compensation (based on external statistics and/or assumptions)
```

Display a specific function in the `Policy`.

```
F(end)
```

```
1×1 Function with properties:

           comment: "MC_EE  step 4: share of hours worked in compensation (based on external statistics and/or assumptions)"
        extensions: [0×1 Extension]
                ID: "743c8700-7db3-4711-a203-e74f7a1d61b4"
              name: "BenCalc"
             order: "27"
        parameters: [11×1 Parameter]
            parent: [1×1 Policy]
             polID: "be1bc6a2-e29f-4750-b6a7-44c061aae718"
           private: "no"
        spineOrder: "8.27"
```

Input Arguments

collapse all

`Policy` — `Policy` class

`class`

A specific policy from the Policy class array.

Data Types: `class`

Properties

collapse all

`comment` — Function comment

`string scalar`

Returns the comment about the function.

Example: `F(end).comment`

`extensions` — Extension class

`class array`

Returns the Extension class array with extensions switches.

Example: `F(end).extensions`

`ID` — Identifier number

`string scalar`

Returns the function identifier number.

Example: `F(end).ID`

`name` — Name of the function

`string scalar`

Returns the name of the function.

Example: `F(end).name`

`order` — Order of the function

`string scalar`

Returns the order of the function.

Example: `F(end).order`

`parameters` — Parameter class

`class array`

Returns the Parameter class array with function parameters.

Example: `F(end).parameters`

`parent` — Policy class.
`class`

Returns the specific Policy class.

Example: `F(end).parent`

`polID` — Identifier number
`string scalar`

Returns the identifier number of the policy.

Example: `F(end).polID`

`private` — Access type
`string scalar`

Returns the access type of the function.

Example: `F(end).private`

`spineOrder` — Spine order of the function
`string scalar`

Returns the spine order of the function.

Example: `F(end).spineOrder`

Output Arguments

collapse all

`F` — Function class
class array

A class array with the `Policy` default functions.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `Policy` | `Parameter` | `FunctionInSystem` | `info` | `run`

Introduced in R2023a

Resources

FunctionInSystem

Functions in a EUROMOD system-policy.

Syntax

```
F = FunctionInSystem(PolicyInSystem)
```

Description

`F = FunctionInSystem(PolicyInSystem)` create a class array with the functions implemented in a policy at the system level. example

This class contains the functions implemented in a specific EUROMOD policy-system. The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. `name`, `ID`, `order`, etc.).

This class is stored in the property functions of the PolicyInSystem class.

This class stores classes of type ParameterInSystem and Extension.

This class inherits methods and properties from the superclass Function.

Examples

collapse all

Load PolicyInSystem functions

`FunctionInSystem` loads all the functions configured in a `Policy` at the system level by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
F=FunctionInSystem(mod.countries("AT").systems(end).policies(8));
F
```

```
27×1 FunctionInSystem array:

     1: DefConst | on  | ESTAT data: unemployment, pre-COVID input data
     2: DefConst | on  | ESTAT data: unemployment, post-COVID input data
     3: DefConst | on  | ESTAT data: MC for employees & self-employed, pre-COVID input data
     4: DefConst | on  | ESTAT data: MC for employees & self-employed, post-COVID input data
     5: Elig     | on  | eligibility conditions - transition from non-employed to employed
     6: BenCalc  | on  | random allocation (based on ESTAT data) - transition non-employed to employed
     7: Elig     | on  | eligibility conditions - transition from employed to unemployed
     8: BenCalc  | on  | random allocation (based on ESTAT data) - transition employed to unemployed for employees
     9: Elig     | on  | eligibility conditions - transition from self-employed to unemployed
    10: BenCalc  | on  | random allocation - transition from self-employed to unemployed
    11: Elig     | on  | eligibility conditions - yem_a
    12: ArithOp  | on  | define yem_a
    13: Elig     | on  | eligibility conditions - lhw_a
    14: ArithOp  | on  | define lhw_a
    15: BenCalc  | on  | set yemmy_a
    16: Elig     | on  | step 1: eligibility conditions: MC for self-employed
    17: BenCalc  | on  | step 2: random allocation (based on Eurostat statistics)
    18: Elig     | on  | selected in step 2 -> eligible for step 3
    19: BenCalc  | on  | step 3a: random allocation of months in compensation scheme (based on external statistics and/or assumptions)
    20: ArithOp  | on  | step 3b: months out of compensation scheme (based on step 3a)
    21: Elig     | on  | MC_EE step 1: eligibility conditions: MC for employees
    22: BenCalc  | on  | MC_EE  step 2: random allocation (based on ESTAT statistics)
    23: Elig     | on  | MC_EE selected in step 2 -> eligible for step 3
    24: BenCalc  | on  | MC_EE step 3a: random allocation of months in MC (based on ESTAT statistics)
    25: ArithOp  | on  | MC_EE step 3b: months out of compensation scheme (based on step 3a)
    26: Elig     | on  | MC_EE  selected in step 3 -> eligible for step 4
    27: BenCalc  | on  | MC_EE  step 4: share of hours worked in compensation (based on external statistics and/or assumptions)
```

Display a specific function in the `PolicyInSystem`.

```
F(end)
```

```
1×1 FunctionInSystem with properties:

         funID: "743c8700-7db3-4711-a203-e74f7a1d61b4"
        Switch: "on"
         sysID: "064cf5eb-73bd-4fae-a421-ed9781e418c9"
       comment: "MC_EE  step 4: share of hours worked in compensation (based on external statistics and/or assumptions)"
    extensions: [0×1 Extension]
            ID: "064cf5eb-73bd-4fae-a421-ed9781e418c9743c8700-7db3-4711-a203-e74f7a1d61b4"
          name: "BenCalc"
         order: "27"
    parameters: [11×1 ParameterInSystem]
        parent: [1×1 PolicyInSystem]
         polID: "be1bc6a2-e29f-4750-b6a7-44c061aae718"
       private: "no"
    spineOrder: "8.27"
```

Input Arguments

collapse all

`PolicyInSystem` — `PolicyInSystem` class
`class`

A specific policy from the PolicyInSystem class array.

Data Types: `class`

Properties

collapse all

`comment` — Function comment
`string scalar`

Returns the comment about the function.

Example: `F(end).comment`

`extensions` — Extension class
`class array`

Returns the Extension class array with extensions switches.

Example: `F(end).extensions`

`funID` — Identifier number
`string scalar`

Returns the identifier number of the function at the `Country` level.

Example: `F(end).funID`

`ID` — Identifier number
`string scalar`

Returns the function identifier number.

Example: `F(end).ID`

`name` — Name of the function
`string scalar`

Returns the name of the function.

Example: `F(end).name`

`order` — Order of the function
`string scalar`

Returns the order of the function.

Example: `F(end).order`

`parameters` — Parameter class
`class array`

Returns the Parameter class array with function parameters.

Example: `F(end).parameters`

`parent` — PolicyInSystem class.
`class`

Returns the specific PolicyInSystem class.

Example: `F(end).parent`

`polID` — Identifier number
`string scalar`

Returns the identifier number of the policy.

Example: `F(end).polID`

`private` — Access type
`string scalar`

Returns the access type of the function.

Example: `F(end).private`

`spineOrder` — Spine order of the function
`string scalar`

Returns the spine order of the function.

Example: `F(end).spineOrder`

`sysID` — Identifier number
`string scalar`

Returns the identifier number of the `System`.

Example: `F(end).sysID`

`Switch` — Extension switch
`string scalar`

Returns the `Extension` switch action.

Example: `F(end).Switch`

Output Arguments

collapse all

`F` — FunctionInSystem class
class array

A class array with the `PolicyInSystem` default functions. These are the functions configured in a specific `System`.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

> **C/C++ Code Generation**
> Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `System` | `PolicyInSystem` | `ParameterInSystem` | `Function` | `info` | `run`

Introduced in R2023a

Resources ▾

Model

The base class of the EUROMOD Connector Toolbox.

Syntax

```
mod = euromod(model_path)
```

Description

`mod = euromod(model_path)` creates the base class `Model` for the microsimulation model EUROMOD. example

This class is a base class of the EUROMOD Connector Toolbox. To load the model base class use the method euromod.

Examples

collapse all

Load model and display objects

`euromod` loads by default all the countries and extensions configured in EUROMOD.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
mod
```

```
1×1 Model with properties:

    extensions: [11×1 Extension]
    countries: [28×1 Country]
    modelpath: "C:\EUROMOD_RELEASES_I6.0+"
```

Input Arguments

collapse all

`model_path` — Path to EUROMOD files

`character vector | string scalar`

Full path to the root directory of the EUROMOD model files.

Data Types: `char | string`

Properties

collapse all

`countries` — Country class array

`class`

Returns a `Country` class array with the EUROMOD countries.

Example: `mod.countries`

`extensions` — Extension class array
`class`

Returns an `Extension` class array with the `Model` extensions.

Example: `mod.extensions`

`modelpath` — Path to the EUROMOD project.
`string scalar`

Returns the path to the EUROMOD project.

Example: `mod.modelpath`

Functions

expand all

run

Output Arguments

collapse all

`mod` — Model class
class

The base class for the EUROMOD model.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Country` | `System` | `Policy` | `Dataset` | `Function` | `Parameter`

Introduced in R2023a

Resources ▾

Parameter

Parameters set up in a EUROMOD policy-function.

Syntax

```
P = Parameter(Function)
```

Description

`P = Parameter(Function)` create a class array with the parameters of a policy `Function`. example

This class contains the parameters specific to a EUROMOD policy function. The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. `name`, `ID`, `order`, etc.).

This class is stored in the property parameters of the Function class.

This class stores classes of type Extension.

This class is the superclass of the ParameterInSystem class.

Examples

collapse all

Load Function parameters

`Parameter` loads all the parameters configured in a `Function` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
P=Parameter(mod.countries("AT").policies(8).functions(end));
P
```

```
11×1 Parameter array:

    1: Who_Must_Be_Elig |
    2: Comp_Cond        |
    3: Comp_perTU       | 0% of hours
    4: Comp_Cond        |
    5: Comp_perTU       | 15% of hours
    6: Comp_Cond        |
    7: Comp_perTU       | 45% of hours
    8: Comp_Cond        |
    9: Comp_perTU       | 70% of hours
   10: Output_Var       |
   11: TAX_UNIT         |
```

Display a specific parameter in the `Function`.

```
P("Output_Var")
```

```
1×1 Parameter with properties:

        comment: ""
    extensions: [0×1 Extension]
         funID: "743c8700-7db3-4711-a203-e74f7a1d61b4"
         group: ""
            ID: "a1a5f7f9-2cc2-4e79-bb1b-75e12ba6cf9b"
          name: "Output_Var"
         order: "10"
        parent: [1×1 Function]
    spineOrder: "8.27.10"
```

Input Arguments

collapse all

`Function` — `Function` class
`class`

A specific function from the Function class array.

Data Types: `class`

Properties

collapse all

`comment` — Parameter comment
`string scalar`

Returns the comment about the parameter.

Example: `P(end).comment`

`extensions` — Extension class
`class array`

Returns the Extension class array with extensions switches.

Example: `P(end).extensions`

`funID` — Identifier number
`string scalar`

Returns the identifier number of the function.

Example: `P(end).funID`

`group` — Group value
`string scalar`

Returns the parameter group value.

Example: `P(end).group`

`ID` — Identifier number
`string scalar`

Returns the parameter identifier number.

Example: `P(end).ID`

`name` — Name of parameter
`string scalar`

Returns the name of the parameter.

Example: `P(end).name`

`order` — Order of parameter
`string scalar`

Returns the order of the parameter.

Example: `P(end).order`

`parent` — Function class.
`class`

Returns the specific Function class.

Example: `P(end).parent`

`spineOrder` — Spine order
`string scalar`

Returns the spine order of the parameter.

Example: `P(end).spineOrder`

Output Arguments

collapse all

`P` — Parameter class
class array

A class array with the `Function` default parameters.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

> **C/C++ Code Generation**
> Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `System` | `Policy` | `Function` | `ParameterInSystem` | `info` | `run`

Introduced in R2023a

Resources ▾

ParameterInSystem

Parameters set up in a function of the EUROMOD specific system.

Syntax

```
P = ParameterInSystem(FunctionInSystem)
```

Description

example

`P = ParameterInSystem(FunctionInSystem)` create a class array with the parameters defined in a function at the system level.

This class contains the parameters specific to a system in the EUROMOD policy function. The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. `name`, `ID`, `order`, etc.).

This class is stored in the property parameters of the FunctionInSystem class.

This class stores classes of type Extension.

This class inherits methods and properties from the superclass Parameter.

Examples

collapse all

Load FunctionInSystem parameters

`ParameterInSystem` loads all the parameters configured in a `Function` and implemented in a `System` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
P=ParameterInSystem(mod.countries("AT").systems(end).policies(8).functions(end));
P
```

```
11×1 ParameterInSystem array:

    1: Who_Must_Be_Elig |              | one
    2: Comp_Cond        |              | (i_mc_rand_3 <= $sh_0hours_ee)
    3: Comp_perTU       | 0% of hours  | 0
    4: Comp_Cond        |              | (i_mc_rand_3 >$sh_0hours_ee & i_mc_rand_3<= $sh_15hours_ee)
    5: Comp_perTU       | 15% of hours | 0.15
    6: Comp_Cond        |              | (i_mc_rand_3 >$sh_15hours_ee & i_mc_rand_3<= $sh_45hours_ee)
    7: Comp_perTU       | 45% of hours | 0.45
    8: Comp_Cond        |              | (i_mc_rand_3 >$sh_45hours_ee)
    9: Comp_perTU       | 70% of hours | 0.70
   10: Output_Var       |              | lhwsr_s
   11: TAX_UNIT         |              | tu_individual_at
```

Display a specific parameter in the `FunctionInSystem`.

```
P("Output_Var")
```

```
1×1 ParameterInSystem with properties:

        parID: "a1a5f7f9-2cc2-4e79-bb1b-75e12ba6cf9b"
        value: "lhwsr_s"
        sysID: "064cf5eb-73bd-4fae-a421-ed9781e418c9"
      comment: ""
   extensions: [0×1 Extension]
        funID: "743c8700-7db3-4711-a203-e74f7a1d61b4"
        group: ""
           ID: "064cf5eb-73bd-4fae-a421-ed9781e418c9a1a5f7f9-2cc2-4e79-bb1b-75e12ba6cf9b"
         name: "Output_Var"
        order: "10"
       parent: [1×1 FunctionInSystem]
   spineOrder: "8.27.10"
```

Input Arguments

collapse all

`FunctionInSystem` — `FunctionInSystem` class

`class`

A specific function from the FunctionInSystem class array.

Data Types: `class`

Properties

collapse all

`comment` — ParameterInSystem comment

`string scalar`

Returns the comment about the parameter.

Example: `P(end).comment`

`extensions` — Extension class

`class array`

Returns the Extension class array with extensions switches.

Example: `P(end).extensions`

`funID` — Identifier number

`string scalar`

Returns the identifier number of the function at country level.

Example: `P(end).funID`

group — Group value

`string scalar`

Returns the parameter group value.

Example: `P(end).group`

`ID` — Identifier number
`string scalar`

Returns the parameter identifier number.

Example: `P(end).ID`

`name` — Name of parameter
`string scalar`

Returns the name of the parameter.

Example: `P(end).name`

`order` — Order of parameter
`string scalar`

Returns the order of the parameter.

Example: `P(end).order`

`parent` — FunctionInSystem class.
`class`

Returns the specific FunctionInSystem class.

Example: `P(end).parent`

`parID` — Identifier number
`string scalar`

Returns the identifier number of the parameter at `Country` level.

Example: `P(end).parID`

`spineOrder` — Spine order
`string scalar`

Returns the spine order of the parameter.

Example: `P(end).spineOrder`

`sysID` — Identifier number
`string scalar`

Returns the identifier number of the system.

Example: `P(end).sysID`

`value` — value of parameter
`string scalar`

Returns the value of the parameter.

Example: `P(end).value`

Output Arguments

collapse all

`P` — ParameterInSystem class
class array

A class array with the `FunctionInSystem` default parameters.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `System` | `PolicyInSystem` | `FunctionInSystem` | `Parameter` | `info` | `run`

Introduced in R2023a

Resources ▾

Policy

Policies implemented in a EUROMOD country model.

Syntax

```
P = Policy(Country)
```

Description

`P = Policy(Country)` create a class array with the policies at `Country` level. example

This class contains tax-benefit policies implemented in a EUROMOD country model. The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. `name`, `ID`, `order`, etc.).

This class is stored in the property policies of the Country class.

This class stores classes of type Extension and Function.

This class is the superclass of the PolicyInSystem class.

Examples

collapse all

Load Country policies

`Policy` loads all the policies configured in a `Country` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
P=Policy(mod.countries("AT"));
P(1:25)
```

```
25×1 Policy array:

     1: setdefault_at |                             | DEF: SET DEFAULT VALUES
     2: uprate_at     |                             | DEF: UPRATING FACTORS
     3: ConstDef_at   |                             | DEF: CONSTANTS
     4: IlsDef_at     |                             | DEF: STANDARD INCOME CONCEPTS
     5: IlsUDBDef_at  |                             | DEF: UDB INCOME CONCEPTS
     6: IlDef_at      |                             | DEF: NON-STANDARD INCOME CONCEPTS
     7: random_at     |                             | DEF: Defining random assignments
     8: TransLMA_at   |                             | DEF: Modelling labour market transitions (DO NOT SWITCH ON; ONLY WORKS WITH LMA ADD-ON)
     9: tudef_at      |                             | DEF: ASSESSMENT UNITS
    10: yem_at        |  (with switch set for MWA)  | DEF: minimum wage (Minimumgehalt)
    11: neg_at        |                             | DEF: recode negative self-employment values to zero
    12: ysecomp_at    |                             | BEN: COVID income compensation for self-employed (ONLY WORKS WITH LMA ADD-ON)
    13: tscer_at      |  (with switch set for SORESI) | SIC: social insurance contributions employer (Arbeitgeberbeitraege zur Sozialversicherung
    14: tscee_at      |  (with switch set for SORESI) | SIC: social insurance contributions employee (Arbeitnehmerbeitraege zur Sozialversicherun
    15: tscse_at      |  (with switch set for SORESI) | SIC: social insurance contributions self-employed (Beitraege zur Sozialversicherung fuer
    16: bch00_at      |                             | BEN: Main child benefit (Familienbeihilfe)
    17: pch00_at      |                             | BEN: child bonus for pensioners (Kinderzuschuss)
    18: pchcs_at      |                             | BEN: child bonus for civil servant pensioners (Kinderzulage)
    19: bunct_at      |  (with switch set for SORESI) | BEN: unemployment benefit (Arbeitslosengeld) (PART SIMULATED)
    20: pmmtu_at      |  (with switch set for SORESI) | BEN: minimum pension top-up (Ausgleichszulage) - turn on pch00_at for full simulation
    21: pcstu_at      |  (with switch set for SORESI) | BEN: minimum pension top-up for civil servants (Ergaenzungszulage) - turn on pch00_at for
    22: tscpe_at      |  (with switch set for SORESI) | SIC: social insurance contributions pensioner (Beitraege zur Sozialversicherung fuer Rent
    23: bch00_at      |  Reference policy             |
    24: bec_at        |                             | BEN: Climate/anti-inflation bonus & one off health insurance bonus self-employed
    25: tin_at        |                             | TAX: income tax (Einkommenssteuer)
```

Display a specific policy in the `Country`.

```
P(19)
```

```
1×1 Policy with properties:

          comment: "BEN: unemployment benefit (Arbeitslosengeld) (PART SIMULATED)"
        functions: [26×1 Function]
          private: "no"
       extensions: "SORESI web-based model: off"
               ID: "5cdaac09-a47c-46fd-ba2e-055bba9578c0"
             name: "bunct_at"
            order: "19"
           parent: [1×1 Country]
      spineOrder: "19"
```

Input Arguments

collapse all

`Country` — `Country` class
`class`

A specific country from the Country class array.

Data Types: `class`

Properties

collapse all

`comment` — Policy comment
`string scalar`

Returns the comment about the policy.

Example: `P(end).comment`

`extensions` — Extension class
`class array`

Returns the Extension class array with extensions switches.

Example: `P(end).extensions`

`functions` — Function class
`class array`

Returns the Function class array with the policy functions.

Example: `P(end).functions`

`ID` — Identifier number
`string scalar`

Returns the policy identifier number.

Example: `P(end).ID`

`name` — Name of policy
`string scalar`

Returns the name of the policy.

Example: `P(end).name`

`order` — Order of policy
`string scalar`

Returns the order of the policy.

Example: `P(end).order`

`parent` — Country class.
`class`

Returns the specific Country class.

Example: `P(end).parent`

`private` — Access type
`string scalar`

Returns the access type of the policy.

Example: `P(end).private`

`spineOrder` — Spine order
`string scalar`

Returns the spine order of the policy.

Example: `P(end).spineOrder`

Output Arguments

collapse all

`P` — Policy class
class array

A class array with the `Country` default policies.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `System` | `Function` | `Parameter` | `ReferencePolicy` | `PolicyInSystem` | `info` | `run`

Introduced in R2023a

Resources ▾

PolicyInSystem

Policies modeled in a EUROMOD tax-benefit system.

Syntax

```
P = PolicyInSystem(System)
```

Description

`P = PolicyInSystem(System)` create a class array with the policies defined in a tax-benefit `System` model. example

This class contains the policy rules implemented in a EUROMOD specific tax-benefit system. The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. `name`, `ID`, `order`, etc.).

This class is stored in the property policies of the System class.

This class stores classes of type FunctionInSystem and Extension.

This class inherits methods and properties from the superclass Policy.

Examples

collapse all

Load System policies

`PolicyInSystem` loads all the policies configured in a `System` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
P=PolicyInSystem(mod.countries("AT").systems(end));
P(1:25)
```

```
25×1 PolicyInSystem array:

     1: setdefault_at | on                             | DEF: SET DEFAULT VALUES
     2: uprate_at     | on                             | DEF: UPRATING FACTORS
     3: ConstDef_at   | on                             | DEF: CONSTANTS
     4: IlsDef_at     | on                             | DEF: STANDARD INCOME CONCEPTS
     5: IlsUDBDef_at  | on                             | DEF: UDB INCOME CONCEPTS
     6: IlDef_at      | on                             | DEF: NON-STANDARD INCOME CONCEPTS
     7: random_at     | on                             | DEF: Defining random assignments
     8: TransLMA_at   | off                            | DEF: Modelling labour market transitions (DO NOT SWITCH ON; ONLY WORKS WITH LMA ADD-ON)
     9: tudef_at      | on                             | DEF: ASSESSMENT UNITS
    10: yem_at        | off (with switch set for MWA)  | DEF: minimum wage (Minimumgehalt)
    11: neg_at        | on                             | DEF: recode negative self-employment values to zero
    12: ysecomp_at    | on                             | BEN: COVID income compensation for self-employed (ONLY WORKS WITH LMA ADD-ON)
    13: tscer_at      | on (with switch set for SORESI) | SIC: social insurance contributions employer (Arbeitgeberbeitraege zur Sozialversicheru
    14: tscee_at      | on (with switch set for SORESI) | SIC: social insurance contributions employee (Arbeitnehmerbeitraege zur Sozialversicher
    15: tscse_at      | on (with switch set for SORESI) | SIC: social insurance contributions self-employed (Beitraege zur Sozialversicherung fue
    16: bch00_at      | on                             | BEN: Main child benefit (Familienbeihilfe)
    17: pch00_at      | off                            | BEN: child bonus for pensioners (Kinderzuschuss)
    18: pchcs_at      | off                            | BEN: child bonus for civil servant pensioners (Kinderzulage)
    19: bunct_at      | on (with switch set for SORESI) | BEN: unemployment benefit (Arbeitslosengeld) (PART SIMULATED)
    20: pmmtu_at      | on (with switch set for SORESI) | BEN: minimum pension top-up (Ausgleichszulage) - turn on pch00_at for full simulation
    21: pcstu_at      | on (with switch set for SORESI) | BEN: minimum pension top-up for civil servants (Ergaenzungszulage) - turn on pch00_at f
    22: tscpe_at      | on (with switch set for SORESI) | SIC: social insurance contributions pensioner (Beitraege zur Sozialversicherung fuer Re
    23: bch00_at      | on                             |
    24: bec_at        | on                             | BEN: Climate/anti-inflation bonus & one off health insurance bonus self-employed
    25: tin_at        | on                             | TAX: income tax (Einkommenssteuer)
```

Display a specific policy in the `System`.

```
P(19)
```

```
1×1 PolicyInSystem with properties:

         polID: "5cdaac09-a47c-46fd-ba2e-055bba9578c0"
        Switch: "on"
         sysID: "064cf5eb-73bd-4fae-a421-ed9781e418c9"
       comment: "BEN: unemployment benefit (Arbeitslosengeld) (PART SIMULATED)"
     functions: [26×1 FunctionInSystem]
       private: "no"
    extensions: "SORESI web-based model: off"
            ID: "064cf5eb-73bd-4fae-a421-ed9781e418c95cdaac09-a47c-46fd-ba2e-055bba9578c0"
          name: "bunct_at"
         order: "19"
        parent: [1×1 System]
    spineOrder: "19"
```

Input Arguments

collapse all

`System` — `System` class

`class`

A specific system from the System class array.

Data Types: `class`

Properties

collapse all

`comment` — Policy comment

`string scalar`

Returns the comment about the policy.

Example: `P(end).comment`

`extensions` — Extension class

`class array`

Returns the Extension class array with extensions switches.

Example: `P(end).extensions`

`functions` — FunctionInSystem class

`class array`

Returns the FunctionInSystem class array with the policy functions.

Example: `P(end).functions`

`ID` — Identifier number

`string scalar`

Returns the policy identifier number.

Example: `P(end).ID`

`name` — Name of policy

`string scalar`

Returns the name of the policy.

Example: `P(end).name`

`order` — Order of policy

`string scalar`

Returns the order of the policy.

Example: `P(end).order`

`parent` — System class.

`class`

Returns the specific System class.

Example: `P(end).parent`

`polID` — Identifier number

`string scalar`

Returns the identifier number of the Policy at country level.

Example: `P(end).polID`

`private` — Access type
`string scalar`

Returns the access type of the policy.

Example: `P(end).private`

`spineOrder` — Spine order
`string scalar`

Returns the spine order of the policy.

Example: `P(end).spineOrder`

`sysID` — Identifier number
`string scalar`

Returns the identifier number of the `System`.

Example: `P(end).sysID`

`Switch` — Extension switch
`string scalar`

Returns the `Extension` switch action.

Example: `P(end).Switch`

Output Arguments

collapse all

`P` — PolicyInSystem class
class array

A class array with the `System` default policies.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `System` | `FunctionInSystem` | `ParameterInSystem` | `ReferencePolicy` | `Policy` | `info` | `run`

Introduced in R2023a

Resources ▾

ReferencePolicy

Reference policies modeled in a EUROMOD country.

Syntax

```
P = ReferencePolicy(Country)
```

Description

`P = ReferencePolicy(Country)` create a class array with the reference policies of a `Country`. example

This class contains the reference policies implemented in a EUROMOD country model. The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. `name`, `ID`, `order`, etc.).

This class is stored in the property policies of the Country class.

This class stores classes of type Extension.

Examples

collapse all

Load Country reference policies

`ReferencePolicy` loads all the reference policies configured in a `Country` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
P=ReferencePolicy(mod.countries("AT"));
P
```

```
25×1 ReferencePolicy array:

    1: bch00_at | BEN: Main child benefit (Familienbeihilfe)
    2: tscee_at | SIC: social insurance contributions employee (Arbeitnehmerbeitraege zur Sozialversicherung)
    3: tin_at   | TAX: income tax (Einkommenssteuer)
    4: bunct_at | BEN: unemployment benefit (Arbeitslosengeld) (PART SIMULATED)
    5: bunnc_at | BEN: Unemployment assistance (Notstandshilfe)
    6: pmmtu_at | BEN: minimum pension top-up (Ausgleichszulage) - turn on pch00_at for full simulation
    7: pcstu_at | BEN: minimum pension top-up for civil servants (Ergaenzungszulage) - turn on pch00_at for full simulation
    8: tscpe_at | SIC: social insurance contributions pensioner (Beitraege zur Sozialversicherung fuer Rentner)
    9: bmact_at | BEN: Maternity allowance
   10: tin_at   | TAX: income tax (Einkommenssteuer)
   11: bch00_at | BEN: Main child benefit (Familienbeihilfe)
```

Display a specific reference policy in the `Country`.

```
P(2)
```

```
1×1 ReferencePolicy with properties:

          refPolID: "bc68928e-d226-466f-a1be-3523e08e7aa1"
        extensions: [0×1 Extension]
                ID: "bc68928e-d226-466f-a1be-3523e08e7aa1"
              name: "tscee_at"
             order: "27"
            parent: [1×1 Country]
        spineOrder: "27"
```

Input Arguments

collapse all

`Country` — `Country` class

`class`

A specific country from the Country class array.

Data Types: `class`

Properties

collapse all

`extensions` — Extension class
`class array`

Returns the Extension class array with extensions switches.

Example: `P(end).extensions`

`ID` — Identifier number
`string scalar`

Returns the reference policy identifier number.

Example: `P(end).ID`

`name` — Name of policy
`string scalar`

Returns the name of the reference policy.

Example: `P(end).name`

`order` — Order of policy
`string scalar`

Returns the order of the reference policy.

Example: `P(end).order`

`parent` — Country class.
`class`

Returns the specific Country class.

Example: `P(end).parent`

`refPolID` — Identifier number
`string scalar`

Returns the identifier number of the policy at the `Country` level.

Example: `P(end).refPolID`

`spineOrder` — Spine order
`string scalar`

Returns the spine order of the reference policy.

Example: `P(end).spineOrder`

Output Arguments

collapse all

`P` — ReferencePolicy class
class array

A class array with the `Country` default reference policies.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

> **C/C++ Code Generation**
> Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `System` | `Function` | `Parameter` | `Policy` | `PolicyInSystem` | `info` | `run`

Introduced in R2023a

Resources ▾

run

Run simulation of a EUROMOD tax-benefit system.

Syntax

```
S = run(Model,country_id,system_id,data,data_id)
S = run(Country,system_id,data,data_id)
S = run(System,data,data_id)
S = run(___,Name,Value)
```

Description

`S = run(Model,country_id,system_id,data,data_id)` returns simulation results running from the `Model` class. example

`S = run(Country,system_id,data,data_id)` returns simulation results running from the `Country` class. example

`S = run(System,data,data_id)` returns simulation results running from the `System` class. example

`S = run(___,Name,Value)` returns simulation results specifying additional optional input arguments. example

Examples

collapse all

Run simulation from the Model class

Running a simulation from the Model class requires 4 other input arguments: two-letters country code, system name, dataset and dataset name. Below is an example for system *SE_2021* of Sweden:

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
data = readtable("SE_2021_b1.txt");
S=mod.run("SE","SE_2021",data,"SE_2021_b1");
S
```

```
1×1 Simulation with properties:

             outputs: {[21671×240 table]}
            settings: [1×1 struct]
    output_filenames: "se_2021_std"
              errors: [0×1 string]

output 1 : [21671x240 table]
    idhh    idperson    idmother    idfather    idpartner    idorighh    idorigperson    dag    dgn    dec
    200      20001           0           0            0         200           20001       45     1      0
    300      30001           0           0        30002         300           30001       26     1      0
    300      30002           0           0        30001         300           30002       26     0      0
    500      50001           0           0        50002         500           50001       37     1      0
    500      50002           0           0        50001         500           50002       33     0      0
    500      50003       50002       50001            0         500           50003        4     1      1
    ...
```

Run simulation from a Country class

Running a simulation from the Country class requires 3 other input arguments: system name, dataset and dataset name.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
data = readtable("SE_2021_b1.txt");
S=mod.("SE").run("SE_2021",data,"SE_2021_b1");
S
```

```
1×1 Simulation with properties:

                outputs: {[21671×240 table]}
               settings: [1×1 struct]
       output_filenames: "se_2021_std"
                 errors: [0×1 string]
```

Run simulation from a System

Running a simulation from the System class requires 2 other input arguments: dataset and dataset name.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
data = readtable("SE_2021_b1.txt");
S=mod.("SE").("SE_2021").run(data,"SE_2021_b1");
S
```

```
1×1 Simulation with properties:

                outputs: {[21671×240 table]}
               settings: [1×1 struct]
       output_filenames: "se_2021_std"
                 errors: [0×1 string]
```

Run simulation with optional input arguments

Specify optional simulation settings as Name-Value input arguments. Below is an example with the addons option:

Open Live Script

```
S=mod.('SE').('SE_2021').run(data,"SE_2021_b1",addons=["MTR","MTR"]);
```

```
Simulation SE_2021_base_mtr, system se_2021_mtr, data SE_2021, country SE_2021_b1: finished.
```

```
S
```

```
1×1 Simulation with properties:

         outputs: {[21671×246 table]  [21671×39 table]}
        settings: [1×1 struct]
output_filenames: ["se_2021_base_mtr"    "se_2021_mtr"]
          errors: [0×1 string]
```

Input Arguments

collapse all

`Model` — Euromod base class `Model`.

`class`

The `Model` class returned by euromod.

Data Types: `class`

`Country` — Country class

`class`

A `country` from the Country class array.

Data Types: `class`

`System` — System class

`class`

A `system` from the System class array.

Data Types: `class`

`country_id` — Two-letters country code

`string scalar`

Two-letters country code as explained in the Eurostat Glossary:Country codes

Data Types: `string scalar`

`system_id` — System name

`string scalar`

The name of a tax-benefit system as configured in EUROMOD.

Data Types: `string scalar`

`data` — Dataset

`table`

A dataset of type table that respects the EUROMOD names of the variables.

Data Types: `table`

`data_id` — Dataset name

`string scalar`

The name of a dataset as configured in EUROMOD.

Data Types: `string scalar`

Nave-Value Input Arguments

collapse all

`addons` — Country addons

`string array`

Name-Value string array with Addons to be integrated in the spine. *Name* is the name of the addon and *Value* is the name of the system in the Addon.

Example: `addons=["MTR","MTR","LMA","LMA_AT"]`

`constantsToOverwrite` — Constants to overwrite

`cell array`

A cell array defining the constants to overwrite in the simulation. Define the constants as Name-Values string arguments, where the *Name* is a string array with the name and the order of the constant, and the *Value* is the new constant value. If the order is `NaN` pass an emtpy string. Default is [].

Example: `constantsToOverwrite={["$tinna_rate2",""],"0.4"}`

`euro` — Get output in euro

`logical`

If `true`, the monetary variables will be converted to euro for the simulation. Default is `false`.

Example: `euro=true`

`nowarnings` — Suppress warnings

`logical`

If `true`, the warning messages resulting from the simulations will be suppressed. Default is `false`.

Example: `nowarnings=true`

`outputpath` — Directory name

`string`

The path to the output directory. When the output path is provided, there will be an output file generated. Default is "".

Example: `outputpath="C:\EUROMOD_RELEASE\Output"`

`public_compoments_only` — Public components

`logical`

If true, the the model will be on with only the public compoments. Default is false.

Example: `public_compoments_only=true`

`switches` — System class
`string array`

Name-Value string array with extensions to be switched `on` or `off`. The *Name* is the short name of the extension and the *Value* is a logical. Default is [].

Example: `switches=["MWA","false"]`

`verbose` — Print output information
`logical`

If true then information on the output will be printed. Default is true.

Example: `verbose=false`

Output Arguments

collapse all

`S` — Simulation class
class array

Simulation class with simulation results and configuration settings.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

S/S++ Code Generation
Generate S and S++ code using MATLAB® Coder™.

See Also

`Model` | `System` | `Policy` | `Dataset` | `Extension` | `info` | `run`

Introduced in R2023a

Resources ▾

Simulation

Simulation results of a EUROMOD tax-benefit system.

Syntax

```
S = Simulation(obj,settings)
```

Description

`S = Simulation(obj,settings)` create a class with the simulation results of a EUROMOD tax-benefit and other configuration settings. example

This class contains simulation results of a EUROMOD tax-benefit system, generated by the method run().

Examples

collapse all

Simulation with default options

Run the simulation of a EUROMOD tax-benefit system with default optional settings:

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
data = readtable("SE_2021_b1.txt");
S=run(mod,"SE","SE_2021",data,"SE_2021_b1");
S
```

```
1×1 Simulation with properties:

         outputs: {[19308×282 table]}
        settings: [1×1 struct]
output_filenames: "se_2021_std"
          errors: [2×1 string]

output 1 : [19308x282 table]
  idhh   idperson   idmother   idfather   idpartner   idorighh   idorigperson   dag   dgn   dec
   200     20001          0          0           0        200          20001    74     1     0
   300     30001          0          0           0        300          30001    77     0     0
   300     30002      30001          0           0        300          30002    53     1     0
   400     40001          0          0       40002        400          40001    68     0     0
   400     40002          0          0       40001        400          40002    65     1     0
   500     50001          0          0       50002        500          50001    78     1     0
  ...
```

Object S is a Simulation class storing the simulation output returned by function run.

Compute the mean of disposable income:

```
mean(S.outputs{1}{:,'ils_dispy'})
```

```
    5.5285e+03
```

Input Arguments

collapse all

`obj` — C# object

`C# class`

A `C#` object returned from the EUROMOD simulation.

Data Types: `C# class`

`settings` — Configuration settings

`struct`

A `struct` with the simulation configuration settings.

Data Types: `struct`

Properties

collapse all

`outputs` — Simulation output

`cell`

Returns a cell array with the table-type simulation results.

Example: `S.outputs{1}`

`settings` — Configuration settings

`struct`

Returns the configuration settings used in the simulation such as addons, constantsToOverwrite, extensions, ect.

Example: `S.settings`

`output_filenames` — Simulation names

`string`

Returns the file-names of simulation outputs.

Example: `S.output_filenames`

`errors` — Errors and warnings

`string`

Returns the errors and warnings from the simulation run.

Example: `S.errors`

Output Arguments

collapse all

S — Simulation class
class

A class with results from the simulation of a `EUROMOD` tax-benefit system.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `System` | `Policy` | `ReferencePolicy` | `Function` | `Parameter` | `info` | `run`

Introduced in R2023a

Resources ▾

System

The EUROMOD tax-benefit systems implemented in a country.

Syntax

```
S = System(Country)
```

Description

`S = System(Country)` create a class array with the tax-benefit systems of a `Country`. example

This class contains the EUROMOD tax-benefit systems of a specific country. The class elements can be accessed by indexing the class array with an integer, or a string value of any class property (e.g. `name`, `ID`, `year`, etc.).

This class is stored in the property systems of the Country class.

This class stores classes of type DatasetInSystem and PolicyInSystem.

Examples

collapse all

Load Country systems

`System` loads all the systems configured in a `Country` by default.

Open Live Script

```
mod = euromod("C:\EUROMOD_RELEASES_I6.0+");
S=System(mod.countries("AT"));
S
```

```
17×1 System array:
     1: AT_2007
     2: AT_2008
     3: AT_2009
     4: AT_2010
     5: AT_2011
     6: AT_2012
     7: AT_2013
     8: AT_2014
     9: AT_2015
    10: AT_2016
    11: AT_2017
    12: AT_2018
    13: AT_2019
    14: AT_2020
    15: AT_2021
    16: AT_2022
    17: AT_2023
```

Display a specific system in the `Country`.

```
S("AT_2021")
```

```
1×1 System with properties:

bestmatchDatasets: "AT_2021_b1"
          comment: ""
   currencyOutput: "euro"
    currencyParam: "euro"
         datasets: "training_data, AT_2019_b2, AT_2021_hhot, AT_2020_b2, AT_2021_b1"
               ID: "ca63e8ca-db68-4f4c-9b90-b5717e7a3776"
      headDefInc: "ils_origy"
             name: "AT_2021"
           parent: [1×1 Country]
         policies: [54×1 PolicyInSystem]
          private: "no"
            order: "16"
             year: "2021"
```

Input Arguments

collapse all

Country — Country class
`class`

A specific country from the Country class array.

Data Types: `class`

Properties

collapse all

bestmatchDatasets — DatasetInSystem class
`class array`

Returns a DatasetInSystem class with best-match datasets for the system.

Example: `S(end).bestmatchDatasets`

comment — System comment
`string scalar`

Returns the comment specific to the system.

Example: `S(end).comment`

currencyOutput — Currency of output
`string scalar`

Returns the currency of the simulation results.

Example: `S(end).currencyOutput`

`currencyParam` — Currency of values

`string scalar`

Returns the currency of the monetary parameters in the system.

Example: `S(end).currencyParam`

`datasets` — DatasetInSystem class

`class array`

Returns a DatasetInSystem class with the datasets available for a system.

Example: `S(end).datasets`

`ID` — Identifier number

`string scalar`

Returns the identifier number of the system.

Example: `S(end).ID`

`headDefInc` — Income definition

`string scalar`

Returns the main income definition.

Example: `S(end).headDefInc`

`name` — System name

`string scalar`

Returns the name of the system.

Example: `S(end).name`

`order` — Order

`string scalar`

Returns the order of the system in the spine.

Example: `S(end).order`

`parent` — Country class.

`class`

Returns the specific Country class.

Example: `S(end).parent`

`policies` — PolicyInSystem class
`class array`

Returns a PolicyInSystem class with system policies.

Example: `S(end).policies`

`private` — Access type
`string scalar`

Returns the access type of the system.

Example: `S(end).private`

`year` — System year
`string scalar`

Returns the year of a system.

Example: `S(end).year`

Object Functions

expand all

info

run

Output Arguments

collapse all

`S` — System class
class array

A class array with the `Country` default systems.

References

[1] Documentation EUROMOD - Tax-benefit microsimulation model for the European Union.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`Model` | `Country` | `PolicyInSystem` | `FunctionInSystem` | `ParameterInSystem` | `info` | `run`

Introduced in R2023a

EUROPEAN UNION PUBLIC LICENCE v. 1.2
EUPL © the European Union 2007, 2016

This European Union Public Licence (the ‘EUPL’) applies to the Work (as defined below) which is provided under the terms of this Licence. Any use of the Work, other than as authorised under this Licence is prohibited (to the extent such use is covered by a right of the copyright holder of the Work).

The Work is provided under the terms of this Licence when the Licensor (as defined below) has placed the following notice immediately following the copyright notice for the Work:

Licensed under the EUPL

or has expressed by any other means his willingness to license under the EUPL.

1. Definitions

In this Licence, the following terms have the following meaning:

- ‘The Licence’: this Licence.

- ‘The Original Work’: the work or software distributed or communicated by the Licensor under this Licence, available as Source Code and also as Executable Code as the case may be.

- ‘Derivative Works’: the works or software that could be created by the Licensee, based upon the Original Work or modifications thereof. This Licence does not define the extent of modification or dependence on the Original Work required in order to classify a work as a Derivative Work; this extent is determined by copyright law applicable in the country mentioned in Article 15.

- ‘The Work’: the Original Work or its Derivative Works.

- ‘The Source Code’: the human-readable form of the Work which is the most convenient for people to study and modify.

- ‘The Executable Code’: any code which has generally been compiled and which is meant to be interpreted by a computer as a program.

- ‘The Licensor’: the natural or legal person that distributes or communicates the Work under the Licence.

- ‘Contributor(s)’: any natural or legal person who modifies the Work under the Licence, or otherwise contributes to the creation of a Derivative Work.

- ‘The Licensee’ or ‘You’: any natural or legal person who makes any usage of the Work under the terms of the Licence.

- ‘Distribution’ or ‘Communication’: any act of selling, giving, lending, renting, distributing, communicating, transmitting, or otherwise making available, online or offline, copies of the Work or providing access to its essential functionalities at the disposal of any other natural or legal person.

2. Scope of the rights granted by the Licence

The Licensor hereby grants You a worldwide, royalty-free, non-exclusive, sublicensable licence to do the following, for the duration of copyright vested in the Original Work:

- use the Work in any circumstance and for all usage,
- reproduce the Work,
- modify the Work, and make Derivative Works based upon the Work,
- communicate to the public, including the right to make available or display the Work or copies thereof to the public and perform publicly, as the case may be, the Work,
- distribute the Work or copies thereof,
- lend and rent the Work or copies thereof,

- sublicense rights in the Work or copies thereof.

Those rights can be exercised on any media, supports and formats, whether now known or later invented, as far as the applicable law permits so.

In the countries where moral rights apply, the Licensor waives his right to exercise his moral right to the extent allowed by law in order to make effective the licence of the economic rights here above listed.

The Licensor grants to the Licensee royalty-free, non-exclusive usage rights to any patents held by the Licensor, to the extent necessary to make use of the rights granted on the Work under this Licence.

3. Communication of the Source Code

The Licensor may provide the Work either in its Source Code form, or as Executable Code. If the Work is provided as Executable Code, the Licensor provides in addition a machine-readable copy of the Source Code of the Work along with each copy of the Work that the Licensor distributes or indicates, in a notice following the copyright notice attached to the Work, a repository where the Source Code is easily and freely accessible for as long as the Licensor continues to distribute or communicate the Work.

4. Limitations on copyright

Nothing in this Licence is intended to deprive the Licensee of the benefits from any exception or limitation to the exclusive rights of the rights owners in the Work, of the exhaustion of those rights or of other applicable limitations thereto.

5. Obligations of the Licensee

The grant of the rights mentioned above is subject to some restrictions and obligations imposed on the Licensee. Those obligations are the following:

Attribution right: The Licensee shall keep intact all copyright, patent or trademarks notices and all notices that refer to the Licence and to the disclaimer of warranties. The Licensee must include a copy of such notices and a copy of the Licence with every copy of the Work he/she distributes or communicates. The Licensee must cause any Derivative Work to carry prominent notices stating that the Work has been modified and the date of modification.

Copyleft clause: If the Licensee distributes or communicates copies of the Original Works or Derivative Works, this Distribution or Communication will be done under the terms of this Licence or of a later version of this Licence unless the Original Work is expressly distributed only under this version of the Licence — for example by communicating ‘EUPL v. 1.2 only’. The Licensee (becoming Licensor) cannot offer or impose any additional terms or conditions on the Work or Derivative Work that alter or restrict the terms of the Licence.

Compatibility clause: If the Licensee Distributes or Communicates Derivative Works or copies thereof based upon both the Work and another work licensed under a Compatible Licence, this Distribution or Communication can be done under the terms of this Compatible Licence. For the sake of this clause, ‘Compatible Licence’ refers to the licences listed in the appendix attached to this Licence. Should the Licensee's obligations under the Compatible Licence conflict with his/her obligations under this Licence, the obligations of the Compatible Licence shall prevail.

Provision of Source Code: When distributing or communicating copies of the Work, the Licensee will provide a machine-readable copy of the Source Code or indicate a repository where this Source will be easily and freely available for as long as the Licensee continues to distribute or communicate the Work.

Legal Protection: This Licence does not grant permission to use the trade names, trademarks, service marks, or names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the copyright notice.

6. Chain of Authorship

The original Licensor warrants that the copyright in the Original Work granted hereunder is owned by him/her or licensed to him/her and that he/she has the power and authority to grant the Licence.

Each Contributor warrants that the copyright in the modifications he/she brings to the Work are owned by him/her or licensed to him/her and that he/she has the power and authority to grant the Licence.

Each time You accept the Licence, the original Licensor and subsequent Contributors grant You a licence to their contributions to the Work, under the terms of this Licence.

7. Disclaimer of Warranty

The Work is a work in progress, which is continuously improved by numerous Contributors. It is not a finished work and may therefore contain defects or ‘bugs’ inherent to this type of development.

For the above reason, the Work is provided under the Licence on an ‘as is’ basis and without warranties of any kind concerning the Work, including without limitation merchantability, fitness for a particular purpose, absence of defects or errors, accuracy, non-infringement of intellectual property rights other than copyright as stated in Article 6 of this Licence.

This disclaimer of warranty is an essential part of the Licence and a condition for the grant of any rights to the Work.

8. Disclaimer of Liability

Except in the cases of wilful misconduct or damages directly caused to natural persons, the Licensor will in no event be liable for any direct or indirect, material or moral, damages of any kind, arising out of the Licence or of the use of the Work, including without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, loss of data or any commercial damage, even if the Licensor has been advised of the possibility of such damage. However, the Licensor will be liable under statutory product liability laws as far such laws apply to the Work.

9. Additional agreements

While distributing the Work, You may choose to conclude an additional agreement, defining obligations or services consistent with this Licence. However, if accepting obligations, You may act only on your own behalf and on your sole responsibility, not on behalf of the original Licensor or any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against such Contributor by the fact You have accepted any warranty or additional liability.

10. Acceptance of the Licence

The provisions of this Licence can be accepted by clicking on an icon ‘I agree’ placed under the bottom of a window displaying the text of this Licence or by affirming consent in any other similar way, in accordance with the rules of applicable law. Clicking on that icon indicates your clear and irrevocable acceptance of this Licence and all of its terms and conditions.

Similarly, you irrevocably accept this Licence and all of its terms and conditions by exercising any rights granted to You by Article 2 of this Licence, such as the use of the Work, the creation by You of a Derivative Work or the Distribution or Communication by You of the Work or copies thereof.

11. Information to the public

In case of any Distribution or Communication of the Work by means of electronic communication by You (for example, by offering to download the Work from a remote location) the distribution channel or media (for example, a website) must at least provide to the public the information requested by the applicable law

regarding the Licensor, the Licence and the way it may be accessible, concluded, stored and reproduced by the Licensee.

12. Termination of the Licence

The Licence and the rights granted hereunder will terminate automatically upon any breach by the Licensee of the terms of the Licence.

Such a termination will not terminate the licences of any person who has received the Work from the Licensee under the Licence, provided such persons remain in full compliance with the Licence.

13. Miscellaneous

Without prejudice of Article 9 above, the Licence represents the complete agreement between the Parties as to the Work.

If any provision of the Licence is invalid or unenforceable under applicable law, this will not affect the validity or enforceability of the Licence as a whole. Such provision will be construed or reformed so as necessary to make it valid and enforceable.

The European Commission may publish other linguistic versions or new versions of this Licence or updated versions of the Appendix, so far this is required and reasonable, without reducing the scope of the rights granted by the Licence. New versions of the Licence will be published with a unique version number.

All linguistic versions of this Licence, approved by the European Commission, have identical value. Parties can take advantage of the linguistic version of their choice.

14. Jurisdiction

Without prejudice to specific agreement between parties,

- any litigation resulting from the interpretation of this License, arising between the European Union institutions, bodies, offices or agencies, as a Licensor, and any Licensee, will be subject to the jurisdiction of the Court of Justice of the European Union, as laid down in article 272 of the Treaty on the Functioning of the European Union,

- any litigation arising between other parties and resulting from the interpretation of this License, will be subject to the exclusive jurisdiction of the competent court where the Licensor resides or conducts its primary business.

15. Applicable Law

Without prejudice to specific agreement between parties,

- this Licence shall be governed by the law of the European Union Member State where the Licensor has his seat, resides or has his registered office,

- this licence shall be governed by Belgian law if the Licensor has no seat, residence or registered office inside a European Union Member State.

Appendix

‘Compatible Licences’ according to Article 5 EUPL are:

- GNU General Public License (GPL) v. 2, v. 3
- GNU Affero General Public License (AGPL) v. 3
- Open Software License (OSL) v. 2.1, v. 3.0
- Eclipse Public License (EPL) v. 1.0
- CeCILL v. 2.0, v. 2.1
- Mozilla Public Licence (MPL) v. 2
- GNU Lesser General Public Licence (LGPL) v. 2.1, v. 3
- Creative Commons Attribution-ShareAlike v. 3.0 Unported (CC BY-SA 3.0) for works other than software

- European Union Public Licence (EUPL) v. 1.1, v. 1.2
- Québec Free and Open-Source Licence — Reciprocity (LiLiQ-R) or Strong Reciprocity (LiLiQ-R+).

The European Commission may update this Appendix to later versions of the above licences without producing a new version of the EUPL, as long as they provide the rights granted in Article 2 of this Licence and protect the covered Source Code from exclusive appropriation.

All other changes or additions to this Appendix require the production of a new EUPL version.

https://joinup.ec.europa.eu/collection/eupl/eupl-text-eupl-12