# Euromod Connector

*Release 0.2.4*

**Belousova Irina, Serruys Hannes**

**Sep 21, 2024**

# CONTENTS

The Euromod Connector for Python 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, calculates the effects of these rules on household income.

CHAPTER

ONE

# CONTENTS

## 1.1 Simulation examples

### 1.1.1 Load model and data

Import the EUROMOD connector Model and the dataset 'PL_2020_b2.txt' for Poland.

```
from euromod import Model
mod=Model(r"C:\EUROMOD_RELEASES_I6.0+")
import pandas as pd
data=pd.read_csv(r"C:\EUROMOD_RELEASES_I6.0+\Input\PL_2020_b2.txt",sep="\t")
```

### 1.1.2 1. Run two systems with default parameters

Run the simulation for the Poland systems 'PL_2021' and 'PL_2022' using the dataset 'PL_2020_b2'.

Checking the best-match datasets for systems 'PL_2021' and 'PL_2022'.

```
print('System    Dataset')
for sys in {'PL_2021', 'PL_2022'}:
    dataset_name = mod['PL']['PL_2022'].bestmatch_datasets[-1].name
    print(sys, ' ', dataset_name)
```

```
System    Dataset
```

```
PL_2021   PL_2020_b2
PL_2022   PL_2020_b2
```

Running multiple simulations in a loop:

```
out=[]
for sysnam in ['PL_2021','PL_2022']:
    out.append(mod['PL'][sysnam].run(data,"PL_2020_b2"))
```

```
Simulation for system PL_2021 with dataset PL_2020_b2 finished.
```

```
Simulation for system PL_2022 with dataset PL_2020_b2 finished.
```

Access the simulation results indexing the `Simulation.outputs` object either with the index position or with the names of the datasets provided in the attribute `Simulation.outputs`:

```
out[1].outputs
```

```
0:             idhh     idperson     idmother  ...     il_bhomx      il_bsamt  \
0          100.0      10001.0          0.0  ...   123.436998  15333.066256
1          100.0      10002.0          0.0  ...     0.000000   6602.816820
2          100.0      10003.0      10002.0  ...     0.000000      0.000000
3          100.0      10004.0      10002.0  ...     0.000000      0.000000
4          200.0      20001.0          0.0  ...   678.998548   1551.658266
...          ...          ...          ...  ...          ...           ...
38637  2047100.0  204710003.0  204710002.0  ...     0.000000      0.000000
38638  2047100.0  204710004.0  204710002.0  ...     0.000000      0.000000
38639  2047200.0  204720001.0          0.0  ...  1160.473138   3394.582426
38640  2047300.0  204730001.0          0.0  ...   395.047916   1476.410557
38641  2047500.0  204750001.0          0.0  ...   716.030976   2816.888295

           il_bsatm
0      15333.066256
1       6602.816820
2          0.000000
3          0.000000
4       1551.658266
...             ...
38637      0.000000
38638      0.000000
38639   3394.582426
38640   1476.410557
38641   2816.888295

[38642 rows x 453 columns]
```

```
out[1].outputs.keys()
```

```
dict_keys(['pl_2022_std.txt'])
```

```
out[1].outputs['pl_2022_std.txt']
```

```
            idhh     idperson     idmother  ...     il_bhomx      il_bsamt  \
0          100.0      10001.0          0.0  ...   123.436998  15333.066256
1          100.0      10002.0          0.0  ...     0.000000   6602.816820
2          100.0      10003.0      10002.0  ...     0.000000      0.000000
3          100.0      10004.0      10002.0  ...     0.000000      0.000000
4          200.0      20001.0          0.0  ...   678.998548   1551.658266
...          ...          ...          ...  ...          ...           ...
38637  2047100.0  204710003.0  204710002.0  ...     0.000000      0.000000
38638  2047100.0  204710004.0  204710002.0  ...     0.000000      0.000000
38639  2047200.0  204720001.0          0.0  ...  1160.473138   3394.582426
38640  2047300.0  204730001.0          0.0  ...   395.047916   1476.410557
38641  2047500.0  204750001.0          0.0  ...   716.030976   2816.888295

           il_bsatm
0      15333.066256
```

(continues on next page)

(continued from previous page)

```
1       6602.816820
2          0.000000
3          0.000000
4       1551.658266
...             ...
38637      0.000000
38638      0.000000
38639   3394.582426
38640   1476.410557
38641   2816.888295

[38642 rows x 453 columns]
```

### 1.1.3 2. Run with changing constant

Run the simulation for the Poland system PL_2022 setting to 10000 the value of the constant '$f_h_cpi' with group number 2022.

```
out=mod['PL']['PL_2022'].run(data,"PL_2020_b2",constantsToOverwrite={("$f_h_cpi","2022"):
↪'10000'})
```

```
Simulation for system PL_2022 with dataset PL_2020_b2 finished.
```

The optional parameter `constantsToOverwrite` specifies which constants to overwrite in the policy spline. `constantsToOverwrite` must be a `dict`, where the keys are `tuples` of two `str` objects: the first string is the name of the constant and the second string is its group number (**Note:** Pass an empty string if the group number is `None`); the values are `str` with the new values of the constants. The default is None.

Attribute `Simulation.constantsToOverwrite` shows the modified constants used in the simulation:

```
out.constantsToOverwrite
```

```
{('$f_h_cpi', '2022'): '10000'}
```

### 1.1.4 3. Run with add-ons

Run the simulation for the Poland system PL_2022 including the Labour Market Adjustment add-on 'LMA'.

```
out =mod['PL']['PL_2022'].run(data,"PL_2020_b2",addons=[("LMA","LMA_PL")])
out
```

```
Simulation for system PL_2022 with dataset PL_2020_b2 finished.
```

```
------------------------------
Simulation
------------------------------
        constantsToOverwrite: {}
        errors: []
        output_filenames: ['pl_2022_lma.txt']
        outputs: Pandas DataFrame of 453 variables and 38642 observations.
```

The optional parameter `addons` is a `list` of EUROMOD Addons to be integrated in the spine . Each item of the `list` is a `tuple` with two `str` objects. The first `str` is the name of the Addon and the second `str` is the name of the system in the Addon to be integrated (typically, it is the name of the Addon _ two-letter country code, e.g. LMA_AT). Available Addons are: LMA, MTR, NRR, TCA. The default is [].

### 1.1.5 4. Run with extensions

Run the simulation for the Poland system PL_2022 switching on the Benefit Take-up Adjustment extension 'BTA'.

```
out =mod['PL']['PL_2022'].run(data,"PL_2020_b2",switches=[("BTA",True)])
out
```

```
Simulation for system PL_2022 with dataset PL_2020_b2 finished.
```

```
------------------------------
Simulation
------------------------------
        constantsToOverwrite: {}
        errors: []
        output_filenames: ['pl_2022_std.txt']
        outputs: Pandas DataFrame of 453 variables and 38642 observations.
```

The optional parameter `switches` must define a `list` of the EUROMOD extensions to be switched on or off in the simulation. Each item in the `list` is a `tuple` with two objects. The first object is a `str` short name of the Extension. The second object is a `boolean`. Available Extensions are: BTA, TCA, FYA, UAA, EPS, PBE, MWA, HHoT_un, WEB, HHoT_ext, HHoT_ncp. The default is [].

## 1.2 User Guide

### 1.2.1 What is the Euromod Conector?

Euromod Conector is a Python library providing tools for running simulations and interacting with the tax-benefit microsimulation model EUROMOD. The fundamental object of the Euromod Connector is the `core.Model` class that nests the EUROMOD country-system models under the attribute `countries`. Each country object is a `core.Country` class that collects in the `systems` attribute the country specific `core.System` classes with the EUROMOD tax-benefit systems. The country and system objects contain other various derived objects, such as datasets, policies, parameters, functions, extensions, and add-ons. The simulation output is returned from the `run` method as a `core.Simulation` class.

---

**Important:** Modifying the objects in the Euromod Connector does not affect the EUROMOD original model that is, the `core.Model` module loads the original EUROMOD model files at each execution.

---

---

**Tip:** The objects that inherit from the `core.Container` class can be indexed in two ways: via an integer or a string being the name of the object.

---

The following indexing conventions apply:

- The objects of the attributes `countries`, `systems`, and `simulations` can be accessed using a single integer or a label. For the country object the label is a two-letter country name, for the system object it

is the system's name, for the simulation object it is the name of the simulation output dataset (Examples: `core.Model.countries['PL']`, `core.Model.countries['PL'].systems['PL_2020']`, `core.Model.countries[3]`, `core.Model.countries[3].systems[10]`).

- The `core.Country` objects can be accessed directly from the model object, i.e. omitting the attribute `countries` (Examples: `core.Model['PL']`, `core.Model[3]` are equivalent to `core.Model.countries[3]`).
- The `core.System` objects can be accessed directly from the country object, i.e. omitting the attribute `systems` (Examples: `core.Model['PL'][0]`, `core.Model[3]['PL_2005']` are equivalent to `core.Model.['PL'].systems['PL_2005']`).

### 1.2.2 Installation

The Euromod Connector can be installed from PyPi using *pip*:

```
$ pip install euromod
```

#### Requirements

The Euromod Connector requires two EUROMOD 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.

**Python version support**

Minimum Python version 3.8 required.

**Windows version support**

Windows 64-bit.

#### Dependencies

The Euromod Connector requires the following dependencies:

| Package | Minimum supported version |
|---|---|
| pandas | 2.0.3 |
| pythonnet | 3.0.2 |

#### Managing Errors

**ModuleNotFoundError or AttributeError**

When the `import` of the Euromod Connector libreries fails, please, uninstall the Python *clr* package and re-install the *pythonnet* package:

```
$ pip uninstall clr
$ pip install pythonnet
```

This error is typically caused by a conflict between the Python *clr* package and the *clr* library of the *pythonnet* package.

**RuntimeError**

Please, perform one of the following tasks:

1) Restart the kernel of the current console window.
2) Open a new console window.
3) Deselect the option *User Module Reloader (UMR)* in the `Tools-> Preferences -> Python Interpreter` (or `Tools -> Console -> Advanced setting`) then press `Apply` and `Ok`, and restart the consol windows.

**Note:** Re-enabling the UMR option has no effect on the console windows that are already open.

This error is typically produced when Python reloads the libraries from the *pythonnet* package.

### 1.2.3 Model

Import the EUROMOD model:

```
from euromod import Model
```

Create an object of the `core.Model` class by passing the path to the EUROMOD project:

```
mod = Model(r"C:\EUROMOD_RELEASES_I6.0+")
```

The model object `mod` has two attributes: the EUROMOD `model_path` defined by the user, and `countries` which instantiates the `core.Country` classes for the EUROMOD default countries.

### 1.2.4 Country

Use `Model.countries` to display the default EUROMOD country objects:

```
mod.countries
```

```
0: AT
1: BE
2: BG
3: CY
```

(continues on next page)

(continued from previous page)

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

Getting the `core.Country` object for Belgium 'BE' (**Note:** The following commands are equivalent):

```
mod.countries['BE']
mod.countries[1]
mod[1]
mod['BE']
```

```
------------------------------
Country
------------------------------
        datasets: 28 elements
        extensions: 13 elements
        local_extensions: Belmod_endo, Belmod_exo
        name: 'BE'
        policies: 42 elements
        systems: 20 elements
```

The attributes of the `core.Country` class store the EUROMOD country-specific objects, such as the available datasets and systems, and the modelled policies and extensions. These objects contain other sub-classes with more specific information about the model.

### Datasets

Attribute `datasets` is a collection of `core.Dataset` objects with all the available datasets for a given country (e.g. Belgium 'BE'):

```
mod["BE"].datasets
```

```
0: BE_2006_a3
1: BE_2007_a3
2: BE_2008_a1
3: training_data
4: BE_2010_a2
5: BE_2012_a5
6: BE_2009_hhot
7: BE_2010_hhot
8: BE_2011_hhot
9: BE_2012_hhot
10: BE_2013_hhot
11: BE_2014_hhot
12: BE_2015_hhot
13: BE_2016_hhot
14: BE_2015_a1
15: BE_2016_a1
16: BE_2017_hhot
17: BE_2018_hhot
18: BE_2019_hhot
19: BE_2017_a4
20: BE_2018_a3
21: BE_2020_hhot
22: BE_2019_c3
23: BE_2021_hhot
24: BE_2020_c2
25: BE_2022_hhot
26: BE_2023_hhot
27: BE_2021_c6
```

Display the general information about a specific dataset by indexing the `datasets` attribute of a country (e.g. Belgium 'BE'):

```
mod["BE"].datasets[-1]
```

```
------------------------------
Dataset
------------------------------
        ID: '2171a46a-7480-41e3-9ee0-7caa85a306c8'
        coicopVersion: ''
        comment: ''
        currency: 'euro'
        decimalSign: '.'
        name: 'BE_2021_c6'
        private: 'no'
        readXVariables: 'no'
        useCommonDefault: 'no'
```

(continues on next page)

(continued from previous page)

```
        yearCollection: '2021'
        yearInc: '2020'
```

### Extensions

Display the extensions modelled for a given country (e.g. Belgium 'BE') using the attribute `local_extensions`:

```
mod.countries[1].local_extensions
```

```
0: BELMOD - Endogenous
1: BELMOD - Exogenous
```

The attribute returns a collection of `core.Extension` objects that can be indexed to get the element-specific information:

```
mod['BE'].local_extensions[0].name
```

```
'BELMOD - Endogenous'
```

### Policies

Use the attribute `policies` to display all the policies for a given country (e.g. Belgium 'BE'):

```
mod["BE"].policies
```

```
0: SetDefault_be          |                                                    |  ␣
↪DEF: Default VALUES
1: uprate_be              |                                                    |  ␣
↪DEF: UPRATING FACTORS
2: ConstDef_be            |  (with switch set for Belmod_endo, Belmod_exo)     |  ␣
↪DEF: Constants
3: ILsDef_be              |                                                    |  ␣
↪DEF: STANDARD INCOME CONCEPTS
4: ILsUDBDef_be           |  (with switch set for Belmod_endo, Belmod_exo)     |  ␣
↪DEF: UDB INCOME CONCEPTS
5: ILDef_be               |                                                    |  ␣
↪DEF: NON-STANDARD INCOME CONCEPTS
6: random_be              |                                                    |  ␣
↪Def: Random number generator
7: TransLMA_be            |                                                    |  ␣
↪DEF: Modelling labour market transitions (DO NOT S ...
8: TUDef_be               |                                                    |  ␣
↪DEF: ASSESSMENT UNITS (OFF for MOTYFF)
9: InitVars_be            |                                                    |  ␣
↪DEF: Initialization of variables
10: yem_be                |  (with switch set for MWA)                         |  ␣
↪DEF: minimum wage  (off in motyff)
11: neg_be                |  (with switch set for Belmod_endo, Belmod_exo)     |  ␣
↪DEF: recode negative income to zero
```

(continues on next page)

(continued from previous page)

```
12: yemcomp_be           |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪BEN: Wage compensation scheme Covid-19
13: ysecomp_be           |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪BEN: Wage compensation scheme Covid-19 (self-emplo ...
14: tscee_be             |                                                       |  ␣
↪SIC: employee (OFF for MOTYFF)
15: tscpe_be             |                                                       |  ␣
↪SIC: pensioners contributions to health and disabi ...
16: tscer_be             |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪SIC: employer (OFF for MOTYFF)
17: tscse_be             |                                                       |  ␣
↪SIC: self-employed
18: tintace_be           |                                                       |  ␣
↪ADMIN TAX: PIT - deduction professional expenses ( ...
19: tinwh_be             |                                                       |  ␣
↪TAX: withholding Income Tax | (not implemented bef ...
20: bmact_be             |  (with switch set for PBE)                            |  ␣
↪BEN: Maternity leave
21: bpact_be             |  (with switch set for PBE)                            |  ␣
↪BEN: Paternity leave benefit
22: bfapl_be             |  (with switch set for PBE)                            |  ␣
↪BEN: Parental leave
23: bun_be               |                                                       |  ␣
↪BEN: Unemployment benefit (PART SIMULATED)
24: byr_be               |                                                       |  ␣
↪BEN: Early Retirement Benefit
25: tprhm_be             |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪TAX: Advance levy on immovable property
26: tintb_be             |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪TAX: PIT - Tax deductions & marital quotient
27: tinna_be             |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪TAX: PIT - Federal Taxes
28: tinrg_be             |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪TAX: PIT - Regional Taxes
29: tinfe_be             |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪TAX: PIT - Fiscal Expenditures
30: tinmu_be             |                                                       |  ␣
↪TAX: PIT - Local Taxes
31: tinkt_be             |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪TAX: Capital Income Tax
32: tsceesp_be           |                                                       |  ␣
↪SIC: special social insurance contribution
33: bchba_be             |                                                       |  ␣
↪BEN: birth allowance
34: bsa_be               |                                                       |  ␣
↪Income support (switch: OFF for MOTYFF, ON for oth ...
35: bch_be               |                                                       |  ␣
↪BEN: child benefit
36: bsaoa_be             |  (with switch set for Belmod_endo, Belmod_exo)        |
↪"BEN: income support for the elderly (TO BE SWITCH ...
37: bed_be               |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪BEN: Study allowances (Flemish and French communit ...
```

(continues on next page)

(continued from previous page)

```
38: bwkrg_be             |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪BEN: Flemish jobbonus
39: tci_be               |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪SIC: Care Insurance Contribution (zorgverzekering) ...
40: output_std_be        |  (with switch set for Belmod_endo, Belmod_exo)        |  ␣
↪DEF: STANDARD OUTPUT INDIVIDUAL LEVEL
41: output_std_hh_be     |                                                       |  ␣
↪DEF: STANDARD OUTPUT HOUSEHOLD LEVEL
```

Get a `core.Policy` specific information by indexing the `policies` attribute (e.g. the employment income policy 'yem_be'):

```
mod["BE"].policies[10]
```

```
------------------------------
Policy
------------------------------
        ID: '923fae10-f0b6-4666-aa2f-ae37bde1d4dc'
        comment: 'DEF: minimum wage  (off in motyff)'
        extensions: ExtensionSwitch Minimum Wage Adjustments: on
        functions: DefConst, Elig, ArithOp, BenCalc, BenCalc, ArithOp, BenCalc, Elig,␣
↪ArithOp
        name: 'yem_be'
        order: '11'
        private: 'no'
        spineOrder: '11'
```

The attributes in the `core.Policy` class contain the information about the policy name, ID, a related comment, as well as objects describing policy functions and extensions.

### Extensions

The attribute `extensions` contains `base.ExtensionSwitch` objects with policy-extension relevant information for a given country (e.g. policy 'yem_be' for Belgium 'BE'):

```
mod["BE"].policies[10].extensions
```

```
0: Minimum Wage Adjustments
```

```
mod["BE"].policies[10].extensions[0]
```

```
------------------------------
ExtensionSwitch
------------------------------
        baseOff: 'false'
        extensionID: '557c232a-9ce6-4808-b52f-ca5e02fe8cf4'
        polID: '923fae10-f0b6-4666-aa2f-ae37bde1d4dc'
```

### Functions

The attribute `functions` stores all the functions related to the specific policy in a country (e.g. policy 'yem_be' for Belgium 'BE'):

```
mod["BE"].policies[10].functions
```

```
0: DefConst    |
1: Elig        |
2: ArithOp     |    monthly wage (corrected for the amounts of months  ...
3: BenCalc     |
4: BenCalc     |
5: ArithOp     |    Adding holiday money to the statutory max
6: BenCalc     |
7: Elig        |
8: ArithOp     |
```

Getting the information about a specific function from the `core.Function` object (e.g. function 'ArithOp' in policy 'yem_be' for Belgium 'BE'):

```
mod["BE"].policies[10].functions[2]
```

```
------------------------------
Function
------------------------------
         ID: 'bb7caeaf-e808-468e-9e1c-de029378ccd2'
         comment: 'monthly wage (corrected for the amounts of months you worked)'
         extensions: 0 elements
         name: 'ArithOp'
         order: '3'
         parameters: Who_Must_Be_Elig, Formula, Output_Var, TAX_UNIT
         polID: '923fae10-f0b6-4666-aa2f-ae37bde1d4dc'
         private: 'no'
         spineOrder: '11.3'
```

Beyond the usual attributes containing the name, identifier and a comment for the function object, the attribute `polID` provides the reference policy identifier from the `core.Policy` object, `parameters` collects the `core.Parameter` objects, and `extensions` attribute includes further modelling information of extensions.

### Parameters

Display all the policy-function related parameters or the specific information about a parameter for a given country (e.g. function 'ArithOp' in policy 'yem_be' for Belgium 'BE'):

```
mod["BE"].policies[10].functions[2].parameters
```

```
0: Who_Must_Be_Elig
1: Formula
2: Output_Var
3: TAX_UNIT
```

Get the specific parameter (e.g. parameter 'Formula' in function 'ArithOp' in policy 'yem_be' for Belgium 'BE'):

```
mod["BE"].policies[10].functions[2].parameters[1]
```

```
------------------------------
Parameter
------------------------------
         ID: '2edfb096-4f7b-48cf-add9-9aed5b3eeab8'
         comment: ''
         extensions: 0 elements
         funID: 'bb7caeaf-e808-468e-9e1c-de029378ccd2'
         group: ''
         name: 'Formula'
         order: '2'
         spineOrder: '11.3.2'
```

### 1.2.5 System

The Euromod Connector stores the EUROMOD tax-benefit systems as `core.System` objects in the attribute `systems` of the `core.Country` class.

Display all the available country systems in a country (e.g. Belgium 'BE'):

```
mod['BE'].systems
```

```
0: BE_2005
1: BE_2006
2: BE_2007
3: BE_2008
4: BE_2009
5: BE_2010
6: BE_2011
7: BE_2012
8: BE_2013
9: BE_2014
10: BE_2015
11: BE_2016
12: BE_2017
13: BE_2018
14: BE_2019
15: BE_2020
16: BE_2021
17: BE_2022
18: BE_2023
19: BE_2023_const
```

Get a specific system object (**Note:** The following commands, returning the system BE_2022 for Belgium, are equivalent):

```
mod[1][17]
mod[1].systems[17]
mod.countries[1].systems[17]
mod.countries['BE'].systems['BE_2022']
```

```
------------------------------
System
------------------------------
         ID: '413c98e1-0fb9-4ff6-8adf-90438cf051b0'
         bestmatch_datasets: BE_2021_c6
         comment: ''
         currencyOutput: 'euro'
         currencyParam: 'euro'
         datasets: training_data, BE_2020_c2, BE_2022_hhot, BE_2021_c6
         headDefInc: 'ils_origrepy'
         name: 'BE_2022'
         order: '26'
         policies: 42 elements
         private: 'no'
         year: '2022'
```

The `core.System` attributes contain the specific system information such as the identifier, the best-match dataset(s),a comment, the currencies of the model parameters and of the simulation output, all the system-specific datasets of type `core.DatasetInSystem` class, the system's name, order, access and the reference year. The `policies` attribute collects the information about the system policies in `core.PolicyInSystem` classes.

### Datasets

Attribute `datasets` stores the `core.DatasetInSystem` objects with all the available datasets for a system (e.g. system 'BE_2022' for Belgium):

```
mod["BE"][17].datasets
```

```
0: training_data     |
1: BE_2020_c2        |
2: BE_2022_hhot      |
3: BE_2021_c6        | best match
```

Getting the information about a specific system-dataset by indexing the `datasets` attribute:

```
mod["BE"][17].datasets[3]
```

```
------------------------------
DatasetInSystem
------------------------------
         ID: '413c98e1-0fb9-4ff6-8adf-90438cf051b02171a46a-7480-41e3-9ee0-7caa85a306c8'
         bestMatch: 'yes'
         coicopVersion: ''
         comment: ''
         currency: 'euro'
         dataID: '2171a46a-7480-41e3-9ee0-7caa85a306c8'
         decimalSign: '.'
         name: 'BE_2021_c6'
         private: 'no'
         readXVariables: 'no'
         sysID: '413c98e1-0fb9-4ff6-8adf-90438cf051b0'
         useCommonDefault: 'no'
```

(continues on next page)

(continued from previous page)

```
        yearCollection: '2021'
        yearInc: '2020'
```

### Policies

The attribute `policies` contains all the system-specific `core.PolicyInSystem` objects describing the policies. Get a specific policy object referring to a country-system model (e.g. the personal income tax policy 'tinmu_be' in system 'BE_2022' for Belgium) by indexing the `policies` attribute:

```
mod["BE"]['BE_2022'].policies[30]
```

```
------------------------------
PolicyInSystem
------------------------------
        ID: '413c98e1-0fb9-4ff6-8adf-90438cf051b07464c9b2-1b1f-416b-acc7-1bd15c72bf56'
        comment: 'TAX: PIT - Local Taxes'
        extensions: 0 elements
        functions: DefConst, BenCalc, ArithOp
        name: 'tinmu_be'
        order: '31'
        polID: '7464c9b2-1b1f-416b-acc7-1bd15c72bf56'
        private: 'no'
        spineOrder: '31'
        switch: 'on'
        sysID: '413c98e1-0fb9-4ff6-8adf-90438cf051b0'
```

With respect to a country class, the policy objects in the system classes store additional information about the identifiers `sysID` and `polID` from, respectively, the `core.System` class and the `core.Policy` class, the policy switch and the order number. The attributes `extensions` and `functions` is a collection of `core.FunctionInSystem` classes.

### Extensions

Display the extensions modelled for a given system-policy (e.g. for policy 'ConstDef_be' in system 'BE_2022' for Belgium 'BE') using the attribute `extensions`:

```
mod["BE"]['BE_2022'].policies[2].extensions
```

```
0: BELMOD - Endogenous
1: BELMOD - Exogenous
```

The attribute returns a collection of `base.ExtensionSwitch` objects that can be indexed to get the element-specific information:

```
mod["BE"]['BE_2022'].policies[2].extensions[0]
```

```
------------------------------
ExtensionSwitch
------------------------------
        baseOff: 'true'
```

(continues on next page)

(continued from previous page)

```
        extensionID: 'af3a504d-4552-47be-b612-a3ff814509b1'
        polID: '4e2539bd-490c-48ce-a4d8-fdd8f4f5fb1e'
```

### Functions

Compared to the country class, the attribute `functions` in the system class, containing `core.FunctionInSystem` objects, additionally shows which functions are used in the simulations of a given system-policy (e.g. policy 'tinmu_be' in system 'BE_2022' for Belgium):

```
mod["BE"]["BE_2022"].policies[30].functions
```

```
0: DefConst     | on (with switch set for Belmod_endo, Belmod_exo)       |
1: BenCalc      | on (with switch set for Belmod_endo, Belmod_exo)       |    Local␣
↪taxes, average per region
2: ArithOp      | on                                                     |    Total PIT␣
↪(Cumulative)
```

Get a specific policy-function object indexing the `functions` attribute (e.g. function 'ArithOp' from policy 'tinmu_be' in system 'BE_2022'):

```
mod["BE"]["BE_2022"].policies[30].functions[2]
```

```
------------------------------
FunctionInSystem
------------------------------
        ID: '413c98e1-0fb9-4ff6-8adf-90438cf051b065bf7c6b-8178-4a20-a31d-71059ea5fce7'
        comment: 'Total PIT (Cumulative)'
        extensions: 0 elements
        funID: '65bf7c6b-8178-4a20-a31d-71059ea5fce7'
        name: 'ArithOp'
        order: '3'
        parameters: Formula, Output_Add_Var, TAX_UNIT
        polID: '7464c9b2-1b1f-416b-acc7-1bd15c72bf56'
        private: 'no'
        spineOrder: '31.3'
        switch: 'on'
        sysID: '413c98e1-0fb9-4ff6-8adf-90438cf051b0'
```

The returned object is a `core.FunctionInSystem` class with some default attributes, such as the name, identifier, extensions, and comment, a series of attributes for the reference identifiers (`funID` from the `core.Function` class, `polID` from the `core.Policy` class, and `sysID` from the `core.System` class), the policy-function switch, and the order number. The attribute `parameters` stores additional modelling information of the system-specific parameters.

#### Parameters

The `core.ParameterInSystem` class, which is stored in the `parameters` attribute, provides modelling information on a specific system-policy-function-parameter element.

Display all the policy-function-specific parameters for a given system (e.g. system 'BE_2022' for the Belgium personal income tax policy 'tinmu_be' in function 'ArithOp') using the attribute `parameters`:

```
mod["BE"]["BE_2022"].policies[30].functions[2].parameters
```

```
0: Formula            | tinmu_s               |    + local taxes
1: Output_Add_Var     | tin_s                 |    = total PIT
2: TAX_UNIT           | tu_individual_be      |
```

Display a specific parameter object by indexing the `parameters` attribute (e.g. the parameter 'Formula' in the function 'ArithOp' from policy 'tinmu_be' in system 'BE_2022'):

```
mod["BE"]["BE_2022"].policies[30].functions[2].parameters[0]
```

```
------------------------------
ParameterInSystem
------------------------------
        ID: '413c98e1-0fb9-4ff6-8adf-90438cf051b0c5d59ade-a653-4fe4-82ea-101324142f95'
        comment: '+ local taxes'
        extensions: 0 elements
        funID: '65bf7c6b-8178-4a20-a31d-71059ea5fce7'
        group: ''
        name: 'Formula'
        order: '1'
        parID: 'c5d59ade-a653-4fe4-82ea-101324142f95'
        spineOrder: '31.3.1'
        sysID: '413c98e1-0fb9-4ff6-8adf-90438cf051b0'
        value: 'tinmu_s'
```

The `core.ParameterInSystem` object contains, additionally to the country object parameter, the order number, the related identifiers from the `core.System` and the `core.Parameter` objects, respectively `sysID` and `parID`, and the parameter `value`.

### 1.2.6 Find objects

The method `find` allows searching for a string pattern in a class attribute of the Euromod Connector. It requires two input parameters: the name of the class attribute (the class name can also be specified using the dot notation), and a string pattern.

Find all the policies containing string 'UPRATING' in the attribute `comment` in country Simpleland 'SL', setting parameter `case_insentive` to `False` (Note that the default is `True`):

```
mod['SL'].policies.find('comment','UPRATING',case_insentive=False)
```

```
0: Uprate_sl    |    DEF: UPRATING FACTORS
```

Find all the functions of a `core.Policy` class of a system, containing string 'on' in the `switch` attribute (e.g. policy 'Uprate_sl' in system 'SL_1996' for country Simpleland 'SL'):

```
mod['SL']['SL_1996'].policies[0].functions.find('switch','on')
```

```
0: Uprate     | on
```

Getting all the policy objects containing 'tax' in the `name` attribute for Poland:

```
mod['PL'].policies.find('name','tax')
```

```
0: tax_hl_fr_pl       |    TAX:Farmer health contribution
1: tax_kt_pl          |    TAX:Lump-sum Capital Income Tax
2: tax_hl_mx_ee_pl    |    TAX:Maximum Health Insurance - employees
3: tax_hl_mx_se_pl    |    TAX:Maximum Health Insurance - self-employed
4: tax_hl_mx_pl       |    TAX:Maximum Health Insurance
5: tax_it_tb_pl       |    TAX:Income Tax Base
6: tax_it_it_pl       |    TAX:Income Tax: Individual Taxation
7: tax_it_lin_pl      |    TAX:Income Tax: Individual Taxation: linear tax
8: tax_it_jt_pl       |    TAX:Income Tax: Joint Taxation
9: tax_it_pl          |    TAX:Income Tax: optimisation
10: tax_hl_pl         |    TAX:Health Insurance
11: tax_ag_pl         |    TAX:Agricultural tax
```

Getting the systems objects containing '2022' in the `name` attribute for Poland:

```
mod['PL'].systems.find('name','2022')
```

```
0: PL_2022
1: PL_2022_const
```

Getting the policy-functions objects containing 'wage' in the `comment` attribute for Poland:

```
mod['PL'].policies[0].functions.find('comment','wage')
```

```
0: DefConst    |    constants for wage compensation scheme
```

Getting the policy functions containing string "BenCalc" in the attribute `name` setting the optional input parameter `return_children=True` (Note that the default is `False`):

```
mod["BE"]["BE_2023"].policies.find("functions.name","Uprate",return_children =True)
```

```
0: Uprate     | on      |
1: Uprate     | on      |    training and hypo data
```

### 1.2.7 Run simulation

Use the `run` method to simulate the EUROMOD tax-benefit systems by passing two required input arguments, a `pandas.DataFrame` dataset and a name of the dataset. For a complete list of parameters please refere to the [API Reference User Guide][].

**Note:** The uprating factors are applied based on the dataset name.

The example below shows how to run a simulation with default optional input parameters for Poland 'PL', tax-benefit system 'PL_2022', using the best match dataset as input data.

Geting the name of the best-match dataset for system 'PL_2022':

```
dataset_id = mod['PL']['PL_2022'].bestmatch_datasets[0].name
dataset_id
```

```
'PL_2020_b2'
```

Load the data as a `pandas.DataFrame` object:

```
import pandas as pd
import os
dataset_path = os.path.join("C:\EUROMOD_RELEASES_I6.0+\Input",dataset_id+".txt")
data = pd.read_csv(dataset_path,sep="\t")
```

Run the simulation providing two input parameters, a `pandas.DataFrame` dataset and a name of the dataset:

```
out=mod.countries['PL'].systems['PL_2022'].run(data,'PL_2020_b2')
```

```
Simulation for system PL_2022 with dataset PL_2020_b2 finished.
```

The simulation run returns a `core.Simulation` class that stores the results as `pandas.DataFrame` objects in the attribute `outputs`:

```
out.outputs[0]
```

```
            idhh     idperson     idmother  ...     il_bhomx      il_bsamt  \
0          100.0      10001.0          0.0  ...   123.436998  15333.066256
1          100.0      10002.0          0.0  ...     0.000000   6602.816820
2          100.0      10003.0      10002.0  ...     0.000000      0.000000
3          100.0      10004.0      10002.0  ...     0.000000      0.000000
4          200.0      20001.0          0.0  ...   678.998548   1551.658266
...          ...          ...          ...  ...          ...           ...
38637  2047100.0  204710003.0  204710002.0  ...     0.000000      0.000000
38638  2047100.0  204710004.0  204710002.0  ...     0.000000      0.000000
38639  2047200.0  204720001.0          0.0  ...  1160.473138   3394.582426
38640  2047300.0  204730001.0          0.0  ...   395.047916   1476.410557
38641  2047500.0  204750001.0          0.0  ...   716.030976   2816.888295

           il_bsatm
0      15333.066256
1       6602.816820
2          0.000000
3          0.000000
4       1551.658266
...             ...
38637      0.000000
38638      0.000000
38639   3394.582426
38640   1476.410557
38641   2816.888295
```

(continues on next page)

(continued from previous page)

```
[38642 rows x 453 columns]
```

## 1.3 API Reference

This reference guide lists the main public objects of the Euromod Connector package. The *euromod.core* module contains most of the public classes of the library. It provides useful functionalities that allow the user to interact with EUROMOD[1] and run simulations. The *euromod.container* module defines a storage class for the model objects accessible by indexing.

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

### 1.3.1 euromod

Below are listed the main public classes of the euromod module.

#### euromod.container

Below are listed the main public classes of the euromod.container module.

Table 1: **Classes**

| | |
|---|---|
| *Container* | This class is a container for objects that allow for indexing and representation in multiple ways: |

**class** euromod.container.**Container**(*idDict=False*)
**This class is a container for objects that allow for indexing and representation in multiple ways:**
**via keys that are the name of the objects**
**or via integer indexing as in a list.**

##### Overview

Table 2: Methods

| | |
|---|---|
| *find*(key, pattern, return_children, case_insentive) | Search for object attributes by pattern. |

##### Methods

**find**(*key*, *pattern*, *return_children=False*, *case_insentive=True*)

Search for object attributes by pattern.

**Parameters**

- **key** (str) – Name of the attribute or the attribute of a child element that you want to look for. One can search child elements by using the dot-notation. E.g.: mod["BE"]["BE_2023"].policies.find("functions.name","BenCalc")

[1] See the documetation for the EUROMOD tax-benefit microsimulation model on the official webpage and in the resources page.

- **pattern** (str) – Pattern that you want to match.
- **return_children** (bool, optional) – When True, the return type will be a *Container* containing elements of the type for which the find method was used When False, the return type will be a *Container* of the elements of the deepest level specified by the pattern key-word. E.g.: mod["BE"]["BE_2023"].policies.find("function) The default is False.
- **case_insentive** (bool, optional) – When false, perform case-insensitive matching. The default is True.

**Returns**
A container of objects that matched the pattern.

**Return type**
*container.Container*

### euromod.core

Below are listed the main public classes of the euromod.core module.

Table 3: **Classes**

| | |
|---|---|
| *Country* | Country-specific EUROMOD tax-benefit model. |
| *Dataset* | Dataset available in a country model. |
| *DatasetInSystem* | Datasets available in a system model. |
| *Extension* | EUROMOD extensions. |
| *ExtensionSwitch* | - |
| *Function* | Functions implemented in a country policy. |
| *FunctionInSystem* | Functions implemented in a policy for a specific system. |
| *Model* | Base class of the Euromod Connector instantiating the microsimulation model |
| *Parameter* | Parameters set up in a function. |
| *ParameterInSystem* | Parameters set up in a function for a specific system. |
| *Policy* | Policy rules modeled in a country. |
| *PolicyInSystem* | Policy rules modeled in a system. |
| *ReferencePolicy* | Object storing the reference policies. |
| *Simulation* | Object storing the simulation results. |
| *System* | A EUROMOD tax-benefit system. |

**class** euromod.core.**Country**(*country: str*, *model: str*)

**Bases: :py:obj:`base.Euromod_Element`**

Country-specific EUROMOD tax-benefit model.

This class instantiates the EUROMOD tax benefit model for a given country. A class instance is automatically generated and stored in the attribute `countries` of the base class *Model*.

This class contains subclasses of type *System*, *Policy*, *Dataset* and *Extension*.

### Example

```
>>> from euromod import Model
>>> mod=Model("C:\EUROMOD_RELEASES_I6.0+")
>>> mod.countries[0]
```

### Overview

Table 4: Attributes

| | |
|---|---|
| *datasets* | A Container with *Dataset* objects. |
| *extensions* | A Container with *Extension* objects. These are the local + model extensions defined. |
| *local_extensions* | A Container with *Extension* objects. These are the local extensions defined for the country. |
| *model* | Returns the base *Model* object. |
| *name* | Two-letters country code. |
| *policies* | A Container with *Policy* objects. |
| *systems* | A Container with *System* objects. |

Table 5: Methods

| | |
|---|---|
| *get_switch_value*(ext_name, dataset_name, sys_name) | **param ext_name**<br>Name of the extension. The default is None. |
| *load_data*(ID_DATASET, PATH_DATA) | Load data as a pandas.DataFrame object. |

### Attributes

**datasets:** *container.Container*[*Dataset*] | None = None

A Container with *Dataset* objects.

**extensions:** *container.Container*[*Extension*] | None = None

A Container with *Extension* objects. These are the local + model extensions defined.

**local_extensions:** *container.Container*[*Extension*] | None = None

A Container with *Extension* objects. These are the local extensions defined for the country.

**model:** *Model*

Returns the base *Model* object.

**Type**
“

**name:** str

Two-letters country code.

**policies:** *container.Container*[*Policy*] | None = None

A Container with *Policy* objects.

**systems:** *container.Container*[*System*] | None = None

A Container with *System* objects.

### Methods

**get_switch_value**(*ext_name: str | None = None*, *dataset_name: str | None = None*, *sys_name: str | None = None*)

**Parameters**

- **ext_name** (str , optional) – Name of the extension. The default is None.
- **dataset_name** (str , optional) – Name of the dataset. The default is None.
- **sys_name** (str , optional) – Name of the system. The default is None.

**Raises**

**KeyError** – Is raised if ext_name, dataset_name or sys_name, but is not configured in the model.

**Returns**

Object of the type *ExtensionSwitch* containing information how the switch is configured. Note that there is only a value returned if the switch is either explicitly 'off' or 'on'. When it's configured as n/a in the model no value will be included.

**Return type**

*core.ExtensionSwitch*

**load_data**(*ID_DATASET*, *PATH_DATA=None*)

Load data as a pandas.DataFrame object.

**Parameters**

- **ID_DATASET** (str) – Name of the dataset excluding extension (Note: must be a *txt* file).
- **PATH_DATA** (str, optional) – Path to the dataset. Default is the PATH_TO_EUROMOD_PROJECT/Input folder.

**Returns**

Dataset is returned as a pandas.DataFrame object.

**Return type**

pandas.DataFrame

**class** euromod.core.**Dataset**(**args*)

**Bases: :py:obj:`base.Euromod_Element`**

Dataset available in a country model.

### Overview

Table 6: Attributes

| | |
|---|---|
| *ID* | Dataset identifier number. |
| *coicopVersion* | COICOP version. |
| *comment* | Comment about the dataset. |
| *currency* | Currency of the monetary values in the dataset. |
| *decimalSign* | Decimal sign |
| *name* | Name of the dataset. |
| *private* | Access type. |
| *readXVariables* | Read variables. |
| *useCommonDefault* | Use default. |
| *yearCollection* | Year of the dataset collection. |
| *yearInc* | Reference year for the income variables. |

### Attributes

**ID:** str
Dataset identifier number.

**coicopVersion:** str = ''
COICOP version.

**comment:** str = ''
Comment about the dataset.

**currency:** str = ''
Currency of the monetary values in the dataset.

**decimalSign:** str = ''
Decimal sign

**name:** str
Name of the dataset.

**private:** str = 'no'
Access type.

**readXVariables:** str = 'no'
Read variables.

**useCommonDefault:** str = 'no'
Use default.

**yearCollection:** str
Year of the dataset collection.

**yearInc:** str
Reference year for the income variables.

**class** euromod.core.**DatasetInSystem**

**Bases: :py:obj:`base.SystemElement`**

Datasets available in a system model.

### Overview

Table 7: Attributes

| | |
|---|---|
| *ID* | Dataset identifier number. |
| *bestMatch* | If yes, the current dataset is a best match for the specific system. |
| *coicopVersion* | COICOP version. |
| *comment* | Comment about the dataset. |
| *currency* | Currency of the monetary values in the dataset. |
| *dataID* | Identifier number of the reference dataset at the country level. |
| *decimalSign* | Decimal sign |
| *name* | Name of the dataset. |
| *private* | Access type. |
| *readXVariables* | Read variables. |
| *sysID* | Identifier number of the reference system. |
| *useCommonDefault* | Use default. |
| *yearCollection* | Year of the dataset collection. |
| *yearInc* | Reference year for the income variables. |

### Attributes

**ID:** str

Dataset identifier number.

**bestMatch:** str

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

**coicopVersion:** str

COICOP version.

**comment:** str

Comment about the dataset.

**currency:** str

Currency of the monetary values in the dataset.

**dataID:** str

Identifier number of the reference dataset at the country level.

**decimalSign:** str

Decimal sign

**name:** str

Name of the dataset.

**private:** str

Access type.

**readXVariables:** str

Read variables.

**sysID:** str

Identifier number of the reference system.

**useCommonDefault:** *str*

Use default.

**yearCollection:** *str*

Year of the dataset collection.

**yearInc:** *str*

Reference year for the income variables.

**class** euromod.core.**Extension**(**arg*)

**Bases: :py:obj:`base.Euromod_Element`**

EUROMOD extensions.

### Overview

Table 8: Attributes

| | |
|---|---|
| *name* | Full name of the extension. |
| *shortName* | Short name of the extension. |

### Attributes

**name:** *str* = **None**

Full name of the extension.

**shortName:** *str* = **None**

Short name of the extension.

**class** euromod.core.**ExtensionSwitch**(*info*, *ctry*)

**Bases: :py:obj:`base.Euromod_Element`**

### Overview

Table 9: Attributes

| | |
|---|---|
| *data_name* | Name of the applicable dataset |
| *extension_name* | Short name of the extension |
| *sys_name* | Name of the applicable system |
| *value* | value of the switch as configured in EUROOMOD. |

### Attributes

**data_name**

Name of the applicable dataset

**extension_name**

Short name of the extension

**sys_name**

Name of the applicable system

**value = ''**

value of the switch as configured in EUROMOD.

**class** euromod.core.**Function**(**arg*)

**Bases: :py:obj:`base.SpineElement`**

Functions implemented in a country policy.

### Overview

Table 10: Attributes

| | |
|---|---|
| *ID* | Identifier number of the function. |
| *comment* | Comment specific to the function. |
| *extensions* | A Container of *Extension* objects in a country. |
| *name* | Name of the function. |
| *order* | Order of the function in the specific spine. |
| *parameters* | A Container of *Parameter* objects in a country. |
| *polID* | Identifier number of the reference policy. |
| *private* | Access type. |
| *spineOrder* | Order of the function in the spine. |

### Attributes

**ID: str**

Identifier number of the function.

**comment: str**

Comment specific to the function.

**extensions: *container.Container*[*Extension*] | None = None**

A Container of *Extension* objects in a country.

**name: str**

Name of the function.

**order: str**

Order of the function in the specific spine.

**parameters: *container.Container*[*Parameter*] | None = None**

A Container of *Parameter* objects in a country.

**polID: str**

Identifier number of the reference policy.

**private: str**

Access type.

**spineOrder: str**

Order of the function in the spine.

**class** euromod.core.**FunctionInSystem**(**arg*)

**Bases: :py:obj:`base.SystemElement`**

Functions implemented in a policy for a specific system.

### Overview

Table 11: Attributes

| | |
|---|---|
| *ID* | Identifier number of the function. |
| *comment* | Comment specific to the function. |
| *extensions* | A `Container` of *Extension* objects in a country. |
| *funID* | Identifier number of the reference function at country level. |
| *name* | Name of the function. |
| *order* | Order of the function in the specific spine. |
| *parameters* | A `Container` with *ParameterInSystem* objects specific to a function. |
| *polID* | Identifier number of the reference policy. |
| *private* | Access type. |
| *spineOrder* | Order of the function in the spine. |
| *switch* | Policy switch action. |
| *sysID* | Identifier number of the reference policy. |

### Attributes

**ID:** str
Identifier number of the function.

**comment:** str
Comment specific to the function.

**extensions:** *container.Container*[*Extension*]
A `Container` of *Extension* objects in a country.

**funID:** str
Identifier number of the reference function at country level.

**name:** str
Name of the function.

**order:** str
Order of the function in the specific spine.

**parameters:** *container.Container*[*ParameterInSystem*] | None = None
A `Container` with *ParameterInSystem* objects specific to a function.

**polID:** str
Identifier number of the reference policy.

**private:** str
Access type.

**spineOrder:** str
Order of the function in the spine.

**switch:** str
Policy switch action.

**sysID:** str
Identifier number of the reference policy.

class euromod.core.**Model**(*model_path: str*)

**Bases: :py:obj:`base.Euromod_Element`**

Base class of the Euromod Connector instantiating the microsimulation model EUROMOD.

**Parameters**

- **model_path** (str) – Path to the EUROMOD project.
- **countries** (str, or Container [ str ], optional) – Countries to load from the project folder. Names must be two-letter country codes, see the Eurostat Glossary:Country codes. If omitted, will load all the available countries in the project folder. Default is None.

**Returns**

A class containing the EUROMOD country models.

**Return type**

*core.Model*

### Example

```
>>> from euromod import Model
>>> mod=Model("C:\EUROMOD_RELEASES_I6.0+")
```

### Overview

Table 12: Attributes

| | |
|---|---|
| *countries* | A Container with *Country* objects. |
| *emPath* | - |
| *errors* | - |
| *extensions* | A Container with *Model* extensions. |
| *model_path* | Path to the EUROMOD project. |

### Attributes

**countries:** *container.Container*[*Country*]

A Container with *Country* objects.

**emPath**

**errors**

**extensions:** *container.Container*[*Extension*]

A Container with *Model* extensions.

**model_path:** str

Path to the EUROMOD project.

**class** euromod.core.**Parameter**(**arg*)

**Bases: :py:obj:`base.SpineElement`**

Parameters set up in a function.

### Overview

Table 13: Attributes

| | |
|---|---|
| *ID* | Identifier number of the parameter. |
| *comment* | Comment specific to the parameter. |
| *extensions* | A Container with *Extension* objects. |
| *funID* | Identifier number of the reference function at country level. |
| *group* | Parameter group value. |
| *name* | Name of the parameter. |
| *order* | Order of the parameter in the specific spine. |
| *spineOrder* | Order of the parameter in the spine. |

### Attributes

**ID: str**

Identifier number of the parameter.

**comment: str**

Comment specific to the parameter.

**extensions: *container.Container*[*Extension*] | None = None**

A Container with *Extension* objects.

**funID: str**

Identifier number of the reference function at country level.

**group: str = ''**

Parameter group value.

**Type**
str

**name: str**

Name of the parameter.

**order: str**

Order of the parameter in the specific spine.

**spineOrder: str**

Order of the parameter in the spine.

**class** euromod.core.**ParameterInSystem**

**Bases: :py:obj:`base.SystemElement`**

Parameters set up in a function for a specific system.

### Overview

Table 14: Attributes

| | |
|---|---|
| *ID* | Identifier number of the parameter. |
| *comment* | Comment specific to the parameter. |
| *extensions* | A `Container` with *Extension* objects. |
| *funID* | Identifier number of the reference function at country level. |
| *group* | Parameter group number. |
| *name* | Name of the parameter. |
| *order* | Order of the parameter in the specific spine. |
| *parID* | Identifier number of the reference parameter at country level. |
| *spineOrder* | Order of the parameter in the spine. |
| *sysID* | Identifier number of the reference system. |
| *value* | Value of the parameter. |

### Attributes

**ID:** str

Identifier number of the parameter.

**comment:** str

Comment specific to the parameter.

**extensions:** list

A `Container` with *Extension* objects.

**funID:** str

Identifier number of the reference function at country level.

**group:** str

Parameter group number.

**Type**

str

**name:** str

Name of the parameter.

**order:** str

Order of the parameter in the specific spine.

**parID:** str

Identifier number of the reference parameter at country level.

**spineOrder:** str

Order of the parameter in the spine.

**sysID:** str

Identifier number of the reference system.

**value:** str

Value of the parameter.

class euromod.core.**Policy**(**arg*)

Bases: :py:obj:`base.SpineElement`

Policy rules modeled in a country.

### Overview

Table 15: Attributes

| | |
|---|---|
| *ID* | Identifier number of the policy. |
| *comment* | Comment specific to the policy. |
| *extensions* | A Container of policy-specific *Extension* objects. |
| *functions* | A Container of policy-specific *Function* objects. |
| *name* | Name of the policy. |
| *order* | Order of the policy in the specific spine. |
| *private* | Access type. Default is 'no'. |
| *spineOrder* | Order of the policy in the spine. |

### Attributes

**ID:** str

Identifier number of the policy.

**comment:** str

Comment specific to the policy.

**extensions:** *container.Container*[*Extension*] | None = None

A Container of policy-specific *Extension* objects.

**functions:** *container.Container*[*Function*] | None = None

A Container of policy-specific *Function* objects.

**name:** str

Name of the policy.

**order:** str

Order of the policy in the specific spine.

**private:** str = 'no'

Access type. Default is 'no'.

**spineOrder:** str

Order of the policy in the spine.

class euromod.core.**PolicyInSystem**(**arg*)

Bases: :py:obj:`base.SystemElement`

Policy rules modeled in a system.

### Overview

Table 16: Attributes

| | |
|---|---|
| *ID* | Identifier number of the policy. |
| *comment* | Comment specific to the policy. |
| *extensions* | A Container of policy-specific *Extension* objects. |
| *functions* | A Container with *FunctionInSystem* objects specific to the system |
| *name* | Name of the policy. |
| *order* | Order of the policy in the specific spine. |
| *polID* | Identifier number of the reference policy at country level. |
| *private* | Access type. Default is 'no'. |
| *spineOrder* | Order of the policy in the spine. |
| *switch* | Policy switch action. |
| *sysID* | Identifier number of the reference system. |

### Attributes

**ID:** str

Identifier number of the policy.

**comment:** str

Comment specific to the policy.

**extensions:** *container.Container*[*Extension*]

A Container of policy-specific *Extension* objects.

**functions:** *container.Container*[*FunctionInSystem*] | None = None

A Container with *FunctionInSystem* objects specific to the system

**name:** str

Name of the policy.

**order:** str

Order of the policy in the specific spine.

**polID:** str

Identifier number of the reference policy at country level.

**private:** str

Access type. Default is 'no'.

**spineOrder:** str

Order of the policy in the spine.

**switch:** str

Policy switch action.

**sysID:** str

Identifier number of the reference system.

**class** euromod.core.**ReferencePolicy**(*info*, *parent*)

**Bases: :py:obj:`base.SpineElement`**

Object storing the reference policies.

### Overview

Table 17: Attributes

| | |
|---|---|
| *extensions* | A Container of reference policy-specific *Extension* objects. |
| *name* | Name of the reference policy. |

### Attributes

**extensions:** *container.Container*[*Extension*] | None = None

A Container of reference policy-specific *Extension* objects.

**name:** str

Name of the reference policy.

**class** euromod.core.**Simulation**(*out*, *constantsToOverwrite*)

**Bases: :py:obj:`base.Euromod_Element`**

Object storing the simulation results.

This is a class containing results from the simulation run and other related configuration information.

### Overview

Table 18: Attributes

| | |
|---|---|
| *constantsToOverwrite* | A dict-type object with user-defined constants. |
| *errors* | A list with errors and warnings from the simulation run. |
| *output_filenames* | A list of file-names of simulation output. |
| *outputs* | A Container with type pandas.DataFrame simulation results. |

### Attributes

**constantsToOverwrite:** dict[tuple(str, str), str]

A dict-type object with user-defined constants.

**errors:** list[str]

A list with errors and warnings from the simulation run.

**output_filenames:** list[str] | [] = []

A list of file-names of simulation output.

**outputs:** *container.Container*[pandas.DataFrame]

A Container with type pandas.DataFrame simulation results. For indexing use an integer or a label from *output_filenames*.

**class** euromod.core.**System**(**arg*)

**Bases: :py:obj:`base.Euromod_Element`**

A EUROMOD tax-benefit system.

This class represents a EUROMOD tax system. Instances of this class are generated automatically when loading a EUROMOD model and are contained in the *systems* Contaiber which is an attribute of the *Country*.

### Example

```
>>> from euromod import Model
>>> mod=Model("C:\EUROMOD_RELEASES_I6.0+")
>>> mod.countries[0].systems[-1]
```

### Overview

Table 19: Attributes

| | |
|---|---|
| *ID* | Identifier number of the system. |
| *bestmatch_datasets* | A Container with best-match *Dataset* objects in the system. |
| *comment* | Comment specific to the system. |
| *currencyOutput* | Currency of the simulation results. |
| *currencyParam* | Currency of the monetary parameters in the system. |
| *datasets* | A Container of *DatasetInSystem* objects in the system. |
| *headDefInc* | Main income definition. |
| *name* | Name of the system. |
| *order* | System order in the spine. |
| *policies* | A Container of *PolicyInSystem* objects in the system. |
| *private* | Access type. |
| *year* | System year. |

Table 20: Methods

| | |
|---|---|
| *run*(data, dataset_id, constantsToOverwrite, verbose, outputpath, addons, switches, nowarnings, euro, public_components_only) | Run the simulation of a EUROMOD tax-benefit system. |

### Attributes

**ID:** str

Identifier number of the system.

**bestmatch_datasets:** *container.Container*[*Dataset*] | None = None

A Container with best-match *Dataset* objects in the system.

**comment:** str

Comment specific to the system.

**currencyOutput:** str

Currency of the simulation results.

**currencyParam:** str

Currency of the monetary parameters in the system.

**datasets:** *container.Container*[*DatasetInSystem*] | None = None

A Container of *DatasetInSystem* objects in the system.

**headDefInc:** str

Main income definition.

**name:** str

Name of the system.

**order:** str

System order in the spine.

**policies:** *container.Container*[*PolicyInSystem*] | None = None

A Container of *PolicyInSystem* objects in the system.

**private:** str

Access type.

**year:** str

System year.

### Methods

**run**(*data: pandas.DataFrame*, *dataset_id: str*, *constantsToOverwrite: Dict[Tuple[str, str], str] | None = None*, *verbose: bool = True*, *outputpath: str = ''*, *addons: List[Tuple[str, str]] = []*, *switches: List[Tuple[str, bool]] = []*, *nowarnings=False*, *euro=False*, *public_components_only=False*)

Run the simulation of a EUROMOD tax-benefit system.

**Parameters**

- **data** (pandas.DataFrame) – input dataframe passed to the EUROMOD model.
- **dataset_id** (str) – ID of the dataset.
- **constantsToOverwrite** (dict [ tuple [ str, str ], str ], optional) – A list of constants to overwrite. Note that the key is a tuple for which the first element is the name of the constant and the second string the groupnumber Default is None.
- **verbose** (bool, optional) – If True then information on the output will be printed. Default is True.
- **outputpath** (str, optional) – When an output path is provided, there will be anoutput file generated. Default is "".
- **addons** (list [ tuple [ str, str ]], optional) – list of addons to be integrated in the spine, where the first element of the tuple is the name of the Addon and the second element is the name of the system in the Addon to be integrated. Default is [].
- **switches** (list [ tuple [ str, bool ]], optional) – list of Extensions to be switched on or of. The first element of the tuple is the short name of the Addon. The second element is a boolean Default is [].
- **nowarnings** (bool, optional) – If True, the warning messages resulting from the simulations will be suppressed. Default is False.
- **euro** (bool, optional) – If True then the monetary variables in the output will be converted to euro. Default value is False.
- **public_compoments_only** (bool, optional) – If True then the the model will be on with only the public compoments. Default value is False.

**Raises**

Exception – Exception when simulation does not finish succesfully, i.e. without errors.

**Returns**

A class containing simulation output and error messages.

**Return type**
*core.Simulation*

### Example

```
>>> # Load the dataset
>>> import pandas as pd
>>> data = pd.read_csv("C:\EUROMOD_RELEASES_I6.0+\Input\sl_demo_v4.txt",sep="
↪")
>>> # Load EUROMOD
>>> from euromod import Model
>>> mod=Model("C:\EUROMOD_RELEASES_I6.0+")
>>> # Run simulation
>>> out=mod.countries['SL'].systems['SL_1996'].run(data,'sl_demo_v4')
```

## 1.4 License

```
                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

# PYTHON MODULE INDEX

## e

# INDEX

## H



## I



## L



## M



## N



## O



## P



## R



## S



## U

## V



## Y