Journal of
open research software

oemof.tabular – Introducing Data Packages for Reproducible Workflows in Energy System Modeling

ISSUES IN RESEARCH SOFTWARE

SIMON HILPERT
STEPHAN GÜNTHER
MARTIN SÖTHE

**Author affiliations can be found in the back matter of this article*

ABSTRACT

The paper describes how Data Packages can be used for creating reproducible workflows in energy system modeling. The presented concept has been implemented in the Python package *oemof.tabular*. The package is designed as an interface to instantiate energy system models from tabular data sources based on the *oemof.solph* library. To implement the data model, *oemof.tabular* extends the Open Energy Modelling Framework (oemof) by facada classes. The developed data model allows users to work with Data Packages and meta data information. The simplified tabular data structure can be used for large energy system models as well as in teaching environments leveraging functionalities of the already widely used *oemof.solph* library.

CORRESPONDING AUTHOR:
Simon Hilpert
Europa Universität Flensburg, Germany
simon.hilpert@uni-flensburg.de

KEYWORDS:
energy system analysis; renewable energy; energy system modeling; open science; reproducible workflows

TO CITE THIS ARTICLE:
Hilpert S, Günther S, Söthe M 2021 *oemof.tabular* – Introducing Data Packages for Reproducible Workflows in Energy System Modeling. *Journal of Open Research Software*, 9: 6. DOI: *https://doi.org/10.5334/jors.320*

1 INTRODUCTION

Analyses of future energy systems are based on tools for complex socio-techno-economic systems. The complexity of these systems increases due to the intermittent supply characteristics of renewable energies which require high temporal and spatial resolution modeling. Additionally, a higher interaction between sectors such as heat, power and transport leads to the need for comprehensive sector coupled approaches. At the same time, a trend towards open source energy (system) models can be observed in the energy system modeling research field [15], as models have been criticized for lack of transparency and reproducibility [17].

For energy system modelers, data handling including input collection, processing and result analysis is one of the most time-consuming tasks. Therefore, open source and open data modeling approaches are put forward as an argument for efficient use of resources [19]. Yet, there is no standardized or broadly used model-agnostic data container in the scientific field of energy system modeling to hold energy system related data. In most cases every software comes with its own logic relating to input-data and output-data of the model. In addition, the decision about how to create the required data sets from raw data sources and the post-processing of result data is often left to the user. Due to these two reasons, re-use of data and more importantly reproducibility of model results is a challenging task, even for experienced modelers.

To improve reproducibility of model results and re-usability of existing data, the following data model description has been developed. Energy system related data is stored in the Data Package format. The complete reproducible workflow from raw-data to final results is described for this data model. The data model has been implemented in the Python package *oemof.tabular* [7] which is based on the Open Energy Modeling Framework (oemof). However, the concept is not restricted to this package, but can be applied with other software as well.

2 BACKGROUND

Oemof is a powerful tool for the modeling of energy systems [8]. Functionalities range from large linear programming (sector coupled) market models [6, 23, 16] to detailed mixed integer heating system [2, 27] or battery models to assess the profitability of power plants in current and future market environments. The underlying concept and its generic implementation allows for this versatile application. It is based on a bipartite graph structure, where nodes are partitioned into buses and components. Most oemof components are of a rather abstract type. For example the `Transformer` class can be used to model different energy system components such as power plants (1 input, 1 output) as well as a heat pump (2 inputs, 1 output) or any other conversion process. To illustrate the concept, ***Figure 1*** shows a `Transformer` connected to different buses (1 input, 2 outputs) to model a combined heat and power (CHP) plant.

The usage of the Python API for this component in *oemof.solph* is shown in ***Figure 2***.

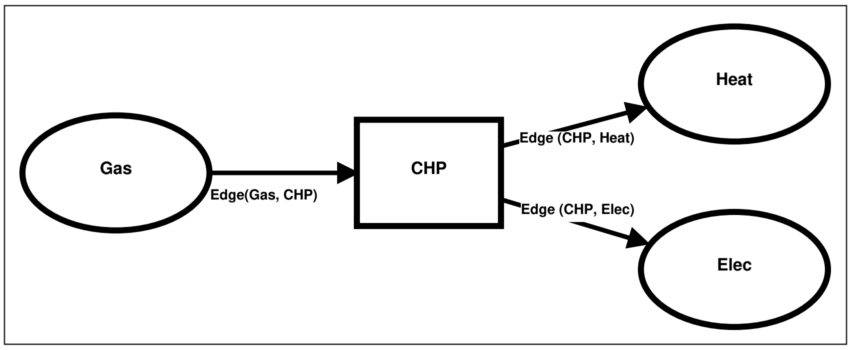

Figure 1 Illustration of a CHP plant model based on the *oemof.solph* `Transformer` class. Nodes are shown as ellipses/squares and edges between the nodes are depicted as arrows.

```
import oemof.solph

Transformer(
    label='CHP',
    inputs={
        Gas: Flow(variable_cost=0.6)},
    outputs={
        Elec: Flow(investment=Investment(ep_costs=50)),
        Heat: Flow(nominal_value=40)}
    conversion_factors={Elec: 0.4, Heat: 0.35})
```

Figure 2 Example of the *oemof.solph* Application Programming Interface (API) for a transformer component with one input and two outputs.

When building energy system models, data is often stored in a tabular data format, for example, CSV files, Excel files or relational databases. However, the design and implementation of a generic tabular data input processing tool for *oemof.solph* has proven to be difficult. One of the reasons is that tables are a flat and two dimensional data structure whereas *oemof.solph*'s API utilizes a high degree of nested objects and data structures. Mapping this nested approach onto flat tables is a non-trivial task.

3 FACADES

Facilitating the task outlined above is one of the functions *oemof.tabular* needs to accomplish. The package was developed in order to allow the user to create an oemof model via tabular data sources. This means that it must also enable her to specify *oemof.solph* components using such flat structures. In *oemof.tabular*, this is done by using the *façade design pattern* first introduced introduced by Gamma et al. in 1994 [5]. The facade design pattern has two main purposes: 1. it provides a simple interface for users to access functionalities of a complex subsystem and 2. it loosens the coupling of consumers of the subsystem's interface with the components of the subsystem.

Therefore, facade classes have the following advantages when viewed in the context of *oemof.tabular*:

- Facades allow instantiation of models from two dimensional data sources as they provide a simplified interface to complex underlying structures.
- The simplified and thus restricted and less generic mathematical representation leads a more transparent modeling approach.
- The simplified interface is easy to use and integrate within the context of teaching and capacity building.
- It also allows building an interface for composed components which are constructed using multiple *oemof.solph* objects.
- Facades can be used with a different back-end, which allows the integration of other energy system modeling frameworks which may not even have to be written in the same programming language.

The implementation

A user of oemof and *oemof.solph* is expected to use instances of classes from a particular class hierarchy to build a model. Thus, facades are integrated into this class hierarchy as a mix-in class: a facade to a specific *oemof.solph* class is created by sub-classing it, mixing in the *Facade* class via cooperative multiple inheritance and then using the general facade methods to simplify the interface of the original *oemof.solph* class.

This allows for a two step approach to build complex components out of simple ones. One can first aggregate a complex subsystem using *composition*, without having to think about simplifying the interface of that system. Simplifying the interface can be done in a second step by creating a facade via *inheritance*.

The *oemof.tabular* package not only provides the facade infrastructure, but also implements a number of facades to regular simple *oemof.solph* components as well as complex compositions of *oemof.solph* components.

Since facades are integrated into *oemof.solph*'s class hierarchy, the classes of all *oemof.tabular* components are sub-classes of *oemof.solph* components, which means that they can freely be mixed with all their more generic parent class objects in a model. In addition, the data model is extendable and could be applied for various model generators, like for example PyPSA [13] or calliope [21]. However, currently the implementation for reading Data Packages is limited to *oemof.tabular* classes. The facade concept as used in *oemof.tabular* has also proved it's applicability by being transported to and used in the *oemof.thermal* [14] package.

The issue of transparency

Model generators such as *oemof.solph* can indeed simplify energy system modeling. However, it is noteworthy that this is a double-edged approach. Simplification for the user always comes with drawbacks as the complexity remains hidden from the user. Depending on the parameters provided, different sets of constraints are created. Nonetheless, resulting mathematical equations are not visible at any stage of the modeling process. Therefore, approaches like *OSeMOSYS* [11] can have a higher level of transparency than other object oriented model generators. As such models or model generators are implemented in a pure algebraic modeling language, every part of the model (variable definitions, constraints, etc.) is clearly and transparently detectable in the source code files. In the case of facades in *oemof.tabular*, mathematical relations of the models and their implementation are hidden by an additional layer of classes. However, since the *oemof.tabular* API is less generic and more restricted than the *oemof.solph* API, the additional layer may actually increase transparency compared to *oemof.solph* components by creating a clear link between input-parameters and the resulting mathematical model.

FACADE EXAMPLE: HYDRO RESERVOIR MODELING

To illustrate the facade concept, subsequently an *oemof.tabular* storage example is compared to the classical *oemof.solph* approach. The *oemof.solph* package provides a `GenericStorage` class to model different storages such as batteries, hot water storages or pumped hydro-electric storages. To model reservoir storages with an inflow and possible spillage, a set of connected *oemof.*

solph components is required. To simplify modeling, the `Reservoir` facade bundles these components and provides a high level API access to a more complex underlying model. ***Figure 3*** provides an illustration of the Reservoir facade.

The facade class itself is a subclass of the `GenericStorage`. However, to allow for a constant inflow into the storage, an additional `Source` object is created.

The reservoir is modeled as a storage with a constant inflow (x denote endogenous variables, c denote exogenous variables):

$$x^{level}(t) = x^{level}(t-1) \cdot (1 - c^{loss\_rate}(t)) + x^{profile}(t) - \frac{x^{flow,out}(t)}{c^{efficiency}(t)} \qquad \forall t \in T \quad (1)$$

$$x^{level}(0) = c^{initial\_storage\_level} \cdot c^{capacity} \quad (2)$$

The inflow is bounded by the exogenous inflow profile. Thus, if the inflow exceeds the maximum capacity of the storage, spillage is possible by setting $x^{profile}(t)$ to lower values.

$$0 \le x^{profile}(t) \le c^{profile}(t) \qquad \forall t \in T \quad (3)$$

The spillage of the reservoir is therefore defined by $c^{profile}(t) - x^{profile}(t)$. Additional constraints apply which have been omitted in the description but can be retrieved from the oemof documentation.

API comparison for the reservoir example

Subsequently, in ***Figure 4***, the Python code to instantiate this component is shown. In comparison to the *oemof.tabular* code, the required *oemof.solph* code differs significantly (see ***Figure 5***). First of all, more objects with a nested set of objects need to be instantiated (Flows, Sources). This nested structure allows for a very flexible modeling approach. However, it creates hurdles for writing a generic data interface to instantiate all these objects, due to the large set of possible combinations. In contrast, the flat structure of the facade arguments allows for a simple interface to tabular data. One additional difference which can be observed is the (energy) specific naming of attributes, for example efficiency, compared to outflow_conversion_factor. As the `Reservoir` class is a subclass of the `GenericStorage` class, some attributes

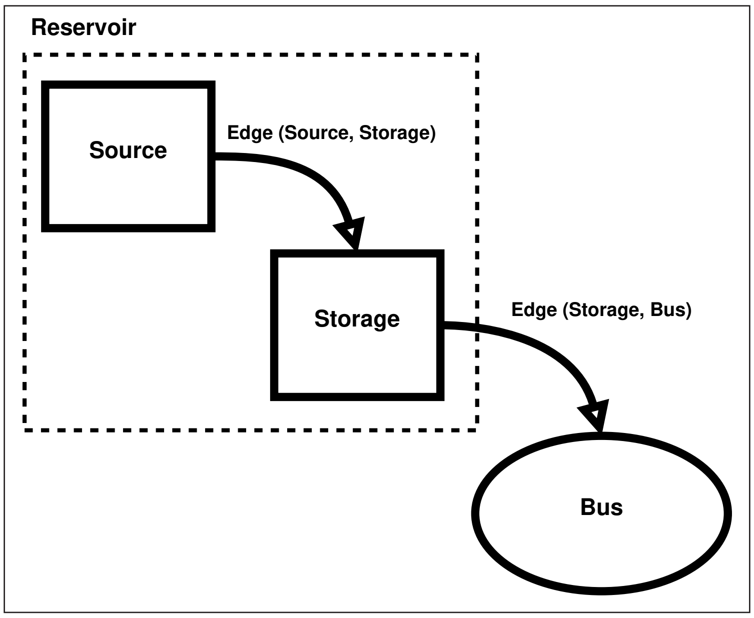

Figure 3 Illustration of a reservoir model in *oemof.tabular*.

```python
from oemof.tabular.facades import Reservoir
from oemof.solph import Bus

bus = Bus("Bus")
rsv = Reservoir(
    label="rsv",
    bus=bus,
    carrier="water",
    tech="reservoir",
    storage_capacity=1000,
    capacity=50,
    profile=[1, 2, 6],
    initial_storage_level=0,
    efficiency=1)
```

Figure 4 API example for an *oemof.tabular* reservoir facade.

```
from oemof.solph import (components, Source, Bus, Flow)

bus = Bus("Bus")
rsv_solph = components.GenericStorage(
    label="rsv-solph",
    nominal_storage_capacity=1000,
    initial_storage_level=0,
    outflow_conversion_factor=1,
    outputs={bus: Flow(nominal_value=50)},
    inputs={})

inflow = Source(
    label="reservoir-inflow",
    outputs={rsv_solph: Flow(nominal_value=1, max=[1, 2, 6],
                                         fixed=False)})
```

Figure 5 API example for a simple *oemof.solph* reservoir model.

of the parent class are also available in the child class (`initial_storage_level`).

Even for comparably small systems, the example underlines the advantages of the approach.

4 DATA PACKAGES

A Data Package is, in its simplest form, not more than a valid JSON [4] file named "*datapackage.json*". The file contains meta data about data resources which can be specified inline in the same file. For more complex cases, data resources are stored in separate files inside the directory containing the "*datapackage.json*" file. The contents of the mentioned JSON file are standardized via the Data Package specification [22]. An example fragment of such a *datapackage.json* JSON file can be seen in ***Figure 7***. The Data Package has been extended by other standards, which further refine the format and contents of the meta data file and the resources to suit different application contexts. Examples of this are Fiscal Data Packages [25], meant to store fiscal data, as well as Tabular Data Packages [26], which refine the original Data Package [22] specification to handle table like data. The latter combines the advantages of databases and spreadsheets with the ubiquity and user-friendliness of CSV files. Tabular Data Packages allow storing type meta data and set primary keys as well as foreign keys across resources, i.e. different CSV files. They are more lightweight than databases and they are both, human readable and easily processable in almost any programming language. In recent years, different European projects in the field of energy system modeling have decided to opt for Data Packages to store model relevant data [18, 10]. Using Data Packages in the correct manner also allows to adhere to the FAIR principle of data handling proposed by Wilkonsen et al. 2016 [29].

In the context of *oemof.tabular*, Data Packages are used to hold information on the topology and parameters of an energy system model instance. At a minimum this includes all exogenous model variables and associated meta data. However, it may also include raw data and scripts for pre- and post processing. On top of the Tabular Data Package structure an structure an energy system specific logic is added, which adds minimal additional constraints on the format of Tabular Data Packages used to specify an *oemof.tabular* model, while still keeping them valid Tabular Data Packages according to the original specification. Therefore, *oemof.tabular* requires the following parts in a Tabular Data Package:

1. a directory named *data* containing at least one sub-folder called *elements*, which may optionally contain a directory called *sequences* and a directory called *geometries* and
2. a valid meta-data `.json` file for the Data Package.

The exemplary folder tree of such a Data Package is depicted in ***Figure 6***.

As stated above, data inside Data Packages is stored in so called resources, which, for a Tabular Data Package, are CSV files. The columns of such resources are referred to as *fields*. Therefore, field names of the resources are equivalent to parameters of the energy system elements and sequences. Connections between components and buses can be defined via foreign keys. These allow linking element fields to fields of other elements stored in other resources. To reference the *name* field of a resource with name *bus* a foreign key can be set within the JSON meta data file using the `forgeinKeys` key as shown in ***Figure 7***.

To distinguish elements and sequences, these two are stored in sub-directories of the data directory. In addition geometrical information can be stored under `data/geometries` in a `.geojson` format. To facilitate the process of creating, processing and calculating a Data Package, *oemof.tabular* offers several functionalities:

```
|-- datapackage
    |-- data
        |-- elements
            |-- demand.csv
            |-- generator.csv
            |-- storage.csv
            |-- bus.csv
        |-- sequences
            |-- volatile-profiles.csv
        |-- geometries
            |-- buses.geojson
    |-- scripts
    |-- datapackage.json
```

Figure 6 Example of an *oemof.tabular* Data Package folder tree.

```
...
"foreignKeys": [
  {
    "fields": "bus",
    "reference": {
      "fields": "name",
      "resource": "bus"
    }
  }
]
```

Figure 7 Setting foreign keys in the JSON meta data file for cross referencing connected components.

- `oemof.tabular.datapackage.building` contains functions to infer meta data, download raw data, read and write elements, sequences etc.
- `oemof.tabular.datapackage.processing` contains functions to process model results, which can be used in the `compute.py` script.
- `oemof.tabular.datapackage.aggregation` allows to aggregate time series to reduce model complexity.

5 REPRODUCIBLE WORKFLOWS

Reproducibility of results is a recurring point of discussions in the energy system modeling community [17, 20]. These discussions have mainly been centered around the availability of source code (open source) and data (open data). Historically, for many prominent models neither the source code nor all input data have been made available. Thanks to new open source developments [8, 11, 13, 21] this has partly changed in recent years (for example the open release of MESSAGEix [12]). However, not all barriers have been removed yet. Firstly, closed models are still being used for research purposes. Secondly, more subtle barriers exist even for open source models. For one of the first open source models, Balmoral [28], a GAMS software license is required, which constitutes a barrier to re-run computations. Another important issue is what can be described as the difference between *practical* and *theoretical* transparency. While for open source models with open data theoretical reproducibility should be possible, practical issues hamper such exercises. First of all, not all necessary information may be given by the respective authors. If provided, complexity of model environments with poor documentation can make any attempt time consuming. In these cases, reproducibility is hardly possible from a practical point of view, even for experienced researchers with domain-specific knowledge.

WORKFLOW DESCRIPTION

To improve reproducibilty of *oemof.tabular*-based research, a structure and workflow is proposed which is based on a the set of ten rules for reproducibility in computational research presented by Sandve et al. 2013 [24]:

1. For every result, keep track of how it was produced
2. Avoid manual data manipulation steps
3. Archive the exact versions of all external programs used
4. Version control all custom scripts
5. Record all intermediate results, when possible in standardized formats
6. For analyses that include randomness, note underlying random seeds
7. Always store raw data behind plots
8. Generate hierarchical analysis output, allowing layers of increasing detail to be inspected
9. Connect textual statements to underlying results
10. Provide public access to scripts, runs, and results

The starting point of this workflow is the folder structure shown in ***Figure 8***.

```
|-- repository
    |-- environment
        |--requirements.txt
            ...
    |-- raw-data
    |-- scenarios
        |--scenario1.toml
        |--scenario2.toml
         ...
    |-- scripts
        |--create-datapackages.py
        |--compute-datapackages.py
            ...
    |-- datapackages
        |-- scenario1
        |-- scenario2
    |-- results
        |--scenario1
            |--input
            |--output
         |-- scenario2
            ...
```

Figure 8 Folder structure for a repository suitable for reproducible workflows.

1. Everything in the `repository` is (if possible) generated by scripts, version controlled, and documented to keep track of every step in result production and avoid manual data manipulation (rule 1, 2). Obviously, the repository is made publicly available (rule 10).
2. The `raw-data` directory contains all input data required to build the input Data Packages for the model. Ideally, raw data sources come with meta data information and open licenses. Unfortunately not all data published comes with such information which hinders reproducibility of workflows. Raw data can also be bundled on remote persistent storages like Zenodo [1], which are suitable for FAIR data distribution.
3. The `scenarios` directory allows to specify different scenarios and describes them in a basic way. The TOML format provides an easy and, if necessary nested structure. In addition to a description, configuration settings for constructing the input Data Packages can be specified in these files. ***Figure 9*** provides an example for a scenario file in the TOML format. This file can be used in the scripts to build Data Packages. Note that the user-specific build-scripts will need to interpret keys and values. Therefore, scenario files in the TOML format do not follow a specific standardized structure, except using the TOML language.
4. The `scripts` directory contains code to construct input Data Packages for scenarios based on the configuration `.toml` files and the raw-data (rule 2). In addition, a script to compute the scenario(s) can be stored there. If possible, raw data can also be downloaded from persistent sources (for example Zenodo) using scripts. Finally, this directory would also contain code for post processing data and for result visualization (rule 7).
5. Results are stored in the `results` directory. One important part is the separation of input and output data. Input data contains model specific exogenous model variables (in this context, *oemof.tabular* Data Packages). The output data directory contains endogenous model variables. Altogether, this step acknowledges rule 5 and 10 of the ten rules.
6. The open license and environment definition in combination with a version control system such as *git* allows to reproduce results on different operating systems (rule 3, 4 and 10).

```
title = "Toy Scenario"
description = "Toy scenario for 3 Nodes"
name = "toy-scenario"

[scenario]
cost = "2030-high"
weather_year = 2011
year = 2030
pv_profiles = "ninja"
onshore_profiles = "emhires"
offshore_profiles = "emhires"

[buses]
electricity = ["DK", "NO", "SE"]
biomass = ["DK", "NO", "SE"]
```

Figure 9 Example *TOML* file with scenario specifications to build input Data Packages.

An example of this workflow has been published for a model-based analysis of the German electricity system [9]. The energy system model covers the German power system with its neighboring countries. Similarly, the workflow has been applied in an analysis for flexibilisation of heat pumps [6].

It should be noted that energy modelers also need to acknowledge energy modeling specific best practices such as proposed by Decarolis et al. [3].

6 CONCLUSION

This paper introduces the application of the facade concept and the usage of Data Packages for the Open Energy System Modeling Framework (oemof). The concept has been implemented in the Python package *oemof.tabular* which is designed as an interface to instantiate energy system models with the *oemof.solph* library from Tabular Data Packages. Using facades can (1) increase transparency by restricting generic components to energy specific components, (2) allow to build composed components and instantiate those from tabular data sources, (3) facilitate the application in teaching and capacity building environments and (4) allow for reproducible workflows. Additionally, the implementation based on the Data Package standard allows to store meta data of the model input data in a standardized way. To enable reproducibility of energy research results a workflow is proposed which is based on scientific literature.

ACKNOWLEDGEMENTS

The authors thank Clemens Wingenbach who took part in the important initial discussions on Data Packages as an input data format. In addition, the Open Modeling Initiative Workshops, with some valuable inspirations by Daniel Huppmann, have been very helpful for the development.

The work has partly been funded by the Federal Ministry of Economic Affairs and Energy, Germany (Grant Number: 03ET6122E). In addition we acknowledge financial support for the article processing charge by the state of Schleswig-Holstein, Germany within the funding program *Open Access Publikationsfond*.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Simon Hilpert *orcid.org/0000-0001-6625-3041*
Europa Universität Flensburg, Germany
Stephan Günther
Otto von Guerike University Magdeburg, Germany
Martin Söthe
Europa Universität Flensburg, Germany

REFERENCES

1. **Zenodo – Research.** Shared. *https://zenodo.org/*.
2. **Boysen C, Kaldemeyer C, Hilpert S, Tuschy, I.** Integration of Flow Temperatures in Unit Commitment Models of Future District Heating Systems. *Energies*. Mar. 2019; 12(6): 1061. DOI: *https://doi.org/10.3390/en12061061*
3. **DeCarolis J, Daly H, Dodds P, Keppo I, Li F, McDowall W, Pye S, Strachan N, Trutnevyte E, Usher W, Winning M, Yeh S, Zeyringer M.** Formalizing best practice for energy system optimization modelling. *Applied Energy*. May 2017; 194: 184–198. DOI: *https://doi.org/10.1016/j.apenergy.2017.03.001*
4. **ECMA.** Standard ECMA-404 The JSON Data Interchange Syntax; 2017.
5. **Gamma E, Helm R, Johnson R, Vlissides J, Booch G.** *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional; 1994.
6. **Hilpert S.** Effects of Decentral Heat Pump Operation on Electricity Storage Requirements in Germany. *Energies*. June 2020; 13(11): 2878. DOI: *https://doi.org/10.3390/en13112878*
7. **Hilpert S, Günther S, Söthe M.** oemof/oemof-tabular, June 2020. original-date: 2018-11-20T16:05:32Z. *https://github.com/oemof/oemof-tabular*.
8. **Hilpert S, Kaldemeyer C, Krien U, Günther S, Wingenbach C, Plessmann G.** The Open Energy Modelling Framework (oemof) – A new approach to facilitate open science in energy system modelling. *Energy Strategy Reviews*. Nov. 2018; 22: 16–25. DOI: *https://doi.org/10.1016/j.esr.2018.07.001*
9. **Hilpert S, Söthe M, Wingenbach C.** *ANGUSII Scenarios: German Energy System 2030*. Zenodo, July 2019. DOI: *https://doi.org/10.5281/zenodo.3714708*
10. **Hotmaps.** Hotmaps Toolbox; 2019. *https://www.hotmaps-project.eu/*.
11. **Howells M, Rogner H, Strachan N, Heaps C, Huntington H, Kypreos S, Hughes A, Silveira S, DeCarolis J, Bazillian M, Roehrl A.** Osemosys: The open source energy modeling system: An introduction to its ethos, structure and development. *Energy Policy*. 2011; 39(10): 5850–5870. Sustainability of biofuels. DOI: *https://doi.org/10.1016/j.enpol.2011.06.033*
12. **Huppmann D, Gidden M, Fricko O, Kolp P, Orthofer C, Pimmer M, Kushin N, Vinca A, Mastrucci A, Riahi K, Krey V.** The MESSAGEix Integrated Assessment Model and the ix modeling platform (ixmp): An open framework for integrated and cross-cutting analysis of energy, climate, the environment, and sustainable development. *Environmental Modelling & Software*. Feb. 2019; 112: 143–156. DOI: *https://doi.org/10.1016/j.envsoft.2018.11.012*
13. **Hörsch J, Hofmann F, Schlachtberger D, Brown T.** PyPSA-Eur: An open optimisation model of the European transmission system. *Energy Strategy Reviews*. Nov. 2018; 22: 207–215. DOI: *https://doi.org/10.1016/j.esr.2018.08.012*
14. **jnnr, FranziPl, Möller C, jakob wo, MaGering, Schönfeldt P, Krien U, Kaldemeyer C, Günther S.** oemof/oemof-thermal: Amazing Absorption (v 0.0.3). July 2020. DOI: *https://doi.org/10.5281/zenodo.3929692*
15. **Lopion P, Markewitz P, Robinius M, Stolten D.** A review of current challenges and trends in energy systems modeling. *Renewable and Sustainable Energy Reviews*. Nov. 2018; 96: 156–166. DOI: *https://doi.org/10.1016/j.rser.2018.07.045*
16. **Maruf MNI.** Sector Coupling in the North Sea Region—A Review on the Energy System Modelling Perspective. *Energies*. Nov. 2019; 12(22): 4298. DOI: *https://doi.org/10.3390/en12224298*
17. **Morrison R.** Energy system modeling: Public transparency, scientific reproducibility, and open development. *Energy Strategy Reviews*. Apr. 2018; 20: 49–63. DOI: *https://doi.org/10.1016/j.esr.2017.12.010*
18. **OPSD.** Open Power System Data – A platform for open data of the European power system. 2019. *https://open-power-system-data.org/*.
19. **Pfenninger S, DeCarolis J, Hirth L, Quoilin S, Staffell I.** The importance of open data and software: Is energy research lagging behind? *Energy Policy*. Feb. 2017; 101: 211–215. DOI: *https://doi.org/10.1016/j.enpol.2016.11.046*
20. **Pfenninger S, Hawkes A, Keirstead J.** Energy systems modeling for twenty-first century energy challenges. *Renewable and Sustainable Energy Reviews*. 2014; 33: 74–86. DOI: *https://doi.org/10.1016/j.rser.2014.02.003*
21. **Pfenninger S, Pickering B.** Calliope: a multi-scale energy systems modelling framework. *Journal of Open Source Software*. Sept. 2018; 3(29): 825. DOI: *https://doi.org/10.21105/joss.00825*
22. **Pollock R, Walsh P.** Data Package; 2007. *https://specs.frictionlessdata.io/data-package/*.
23. **Prina MG, Casalicchio V, Kaldemeyer C, Manzolini G, Moser D, Wanitschke A, Sparber W.** Multi-objective investment optimization for energy system models in high temporal and spatial resolution. *Applied Energy*. Apr. 2020; 264: 114728. DOI: *https://doi.org/10.1016/j.apenergy.2020.114728*
24. **Sandve GK, Nekrutenko A, Taylor J, Hovig E.** Ten Simple Rules for Reproducible Computational Research. *PLOS Computational Biology*. Oct. 2013; 9(10): e1003285. DOI: *https://doi.org/10.1371/journal.pcbi.1003285*

25. **Walsh P, Pollock R, Björgvinsson T, Bennett S, Kariv A, Fowler D.** Fiscal Data Package; 2014. *https://specs.frictionlessdata.io/fiscal-data-package/*.
26. **Wash P, Pollock R, Keegan M.** Tabular Data Package; 2012. *https://specs.frictionlessdata.io/tabular-data-package/*.
27. **Wehkamp S, Schmeling L, Vorspel L, Roelcke F, Windmeier K-L.** District Energy Systems: Challenges and New Tools for Planning and Evaluation. *Energies.* June 2020; 13(11): 2967. DOI: *https://doi.org/10.3390/en13112967*
28. **Wiese F, Bramstoft R, Koduvere H, Pizarro Alonso A, Balyk O, Kirkerud JG, Tveten AG, Bolkesjö TF, Münster M, Ravn H.** Balmorel open source energy system model. *Energy Strategy Reviews.* Apr. 2018; 20: 26–34. DOI: *https://doi.org/10.1016/j.esr.2018.01.003*
29. **Wilkinson MD, Dumontier M, Aalbersberg IJ, Appleton G, Axton M, Baak A, Blomberg N, Boiten J-W, da Silva Santos LB, Bourne PE, Bouwman J, Brookes AJ, Clark T, Crosas M, Dillo I, Dumon O, Edmunds S, Evelo CT, Finkers R, Gonzalez-Beltran A, Gray AJ, Groth P, Goble C, Grethe JS, Heringa J, 't Hoen PA, Hooft R, Kuhn T, Kok R, Kok J, Lusher SJ, Martone ME, Mons A, Packer AL, Persson B, Rocca-Serra P, Roos M, van Schaik R, Sansone S-A, Schultes E, Sengstag T, Slater T, Strawn G, Swertz MA, Thompson M, van der Lei J, van Mulligen E, Velterop J, Waagmeester A, Wittenburg P, Wolstencroft K, Zhao J, Mons B.** The FAIR Guiding Principles for scientific data management and stewardship. Mar. 2016; 3: 160018. DOI: *https://doi.org/10.1038/sdata.2016.18*

TO CITE THIS ARTICLE:
Hilpert S, Günther S, Söthe M 2021 *oemof.tabular* – Introducing Data Packages for Reproducible Workflows in Energy System Modeling. *Journal of Open Research Software,* 9: 6. DOI: *https://doi.org/10.5334/jors.320*

Submitted: 30 January 2020 Accepted: 30 October 2020 Published: 05 May 2021