Article

d2ix: A Model Input-Data Management and Analysis Tool for MESSAGE$_{ix}$

Thomas Zipperle *,† and **Clara Luisa Orthofer** †

Chair of Energy Economy and Application Technology, Department of Electrical and Computer Engineering, Technical University of Munich, 80333 Munich, Germany; clara.orthofer@tum.de

* Correspondence: thomas.zipperle@tum.de

† These authors contributed equally to this work.

Received: 7 March 2019; Accepted: 15 April 2019; Published: 18 April 2019

Abstract: Bottom-up integrated assessment models, like MESSAGE$_{ix}$, depend on the description of the capabilities and limitations of technological, economical and ecological parameters, and their development over long-time horizons. Even small models of a few nodes, technologies and model years require input-data sets involving several hundred thousand data points. Such data sets quickly become incomprehensible, which makes error detection, collaborative working and the interpretation of results challenging, especially for non-self-created models. In response to the resulting need for manageable, comprehensible, and traceable representation of input-data, we developed a Python-based spreadsheet interface (*d2ix*) that enables presentation and editing of model input-data in a concise form. By increasing accessibility and transparency of the model input-data, *d2ix* reduces barriers to entry for new modellers and simplifies collaborative working. This paper describes the methodology and introduces the open-source Python-package *d2ix*. The package is available under the Apache License, Version 2.0 on GitHub.

Keywords: MESSAGE$_{ix}$; reproducibility; collaborative work; open modelling and data; data-handling; integrated assessment modelling; data pre- and post-processing

1. Introduction

The software package described in the following —*d2ix*— is freely available under the Apache License, Version 2.0 on GitHub under: https://github.com/tum-ewk/d2ix.

1.1. *Input-Data-Handling—The Underrated Modelling Challenge*

Technology-based integrated assessment models, such as MESSAGE$_{ix}$ (formerly known as MESSAGE) have a long history in energy and environmental systems modelling [1,2]. Despite having been developed in times of relatively low computing power, over the last forty years these models have grown in line with multiplying computing capacity, expanding models in dimensions such as coverage and detail [3]. Until the 1990s, the models focused on the energy-system only [4]; however, today's energy-engineering-economic-environment optimising models are designed to describe the full extent of energy-system dynamics, including effects such as polluting greenhouse gas emissions, economic development, land and water use and health implications [5,6]. At the same time, rising computing power allows not only increasing coverage but also magnifies the level of detail represented in models, such as the number of model years, nodes, technologies and technology parameters.

While in line with this structural change, big data not only presents a challenge in terms of energy-systems modelling: here too, the amount of input-data has skyrocketed [7]. Today, one technology in MESSAGE$_{ix}$ is described by forty parameters, of which fourteen are defined not only by the installation year of the technology but also the age of the technology. Thus, in even very simple input-data sets (e.g., describing one node over ten model years), each technology is defined by approximately one thousand input parameters, each again defined by up to twelve sets. Therefore, even a small model of one node over ten years has an input-data set per technology of more than twelve-thousand data points, not including the input-data for describing the ecology or economy (Figure 1).

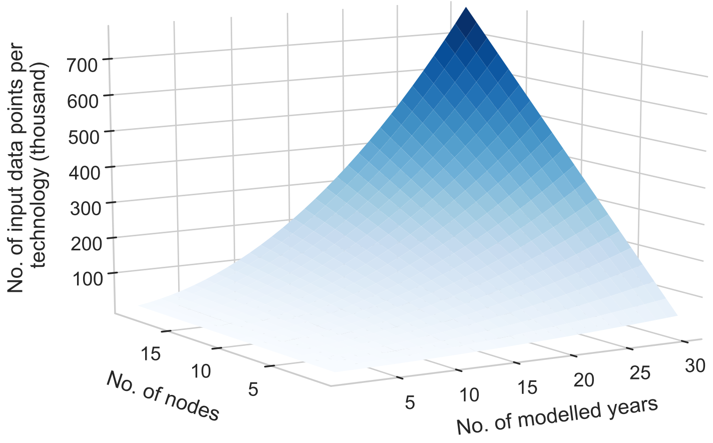


Figure 1. Average number of data points per technology in the input-data set of a MESSAGE$_{ix}$ model in dependence of the number of modelled years and nodes.

1.2. *d2ix—Combining Benefits of Non-Binary and Binary Data Formats*

Currently, most models handle input-data using vast spreadsheets (e.g., MS Excel), csv (comma-separated value) or plain text-files to organise, pre-process and document the model-data. While on the one hand, binary ('higher') formats, such as spreadsheets, provide support with data-handling, (un)intentional changes made to the input files are not trackable and are difficult to retrace. On the other hand, non-binary ('lower') but trackable formats such as csv and text-files lack visual clarity and data-handling support. To ensure transparency in data-handling and reproducibility of model results, the modelling-platform (ixmp), supplies MESSAGE$_{ix}$ users with tools for (i) database communication for version-controlled data management, (ii) a Python/R interface for efficient input-data and results processing and (iii) a web-browser based tool for drag and drop results visualisation [8]. The newly developed 'data to MESSAGE$_{ix}$' (*d2ix*) package adds to this functionality by providing the user with a visually comprehensible overview of the input-data by reducing the dimensions of the input-data set, thereby reducing the number of data points to be handled by the user (Figure 2).

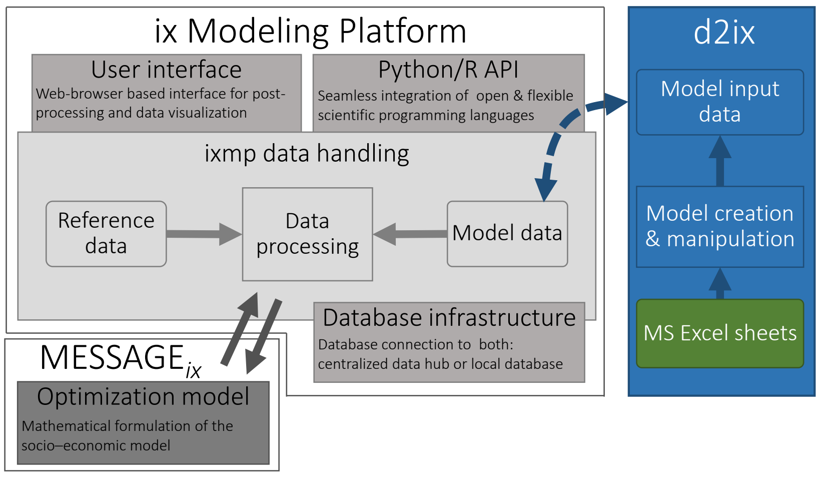
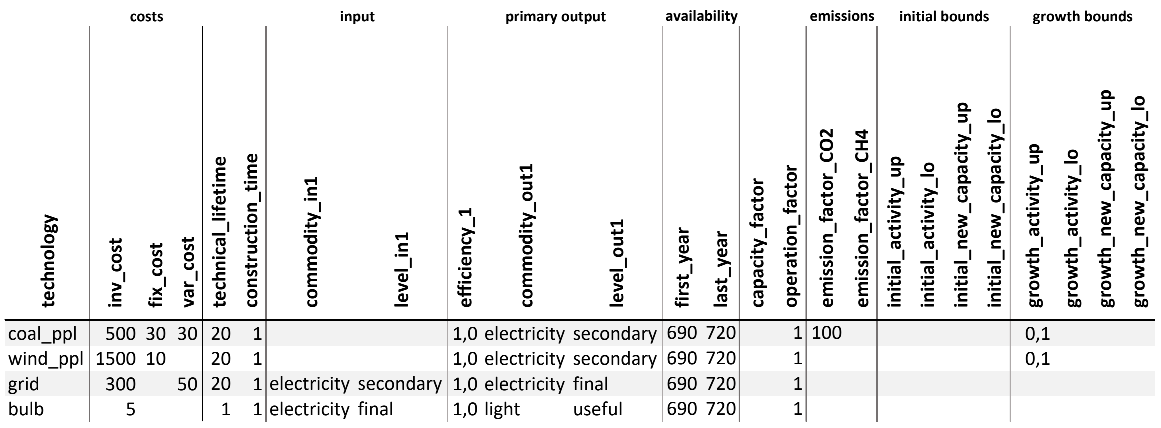

ix Modeling Platform
User interface
Web-browser based interface for post-processing and data visualization
Python/R API
Seamless integration of open & flexible scientific programming languages
ixmp data handling
Reference data
Data processing
Model data
Database infrastructure
Database connection to both: centralized data hub or local database
MESSAGE$_{ix}$
Optimization model
Mathematical formulation of the socio–economic model
d2ix
Model input data
Model creation & manipulation
MS Excel sheets

Figure 2. Integration and interlinkages of *d2ix* to the *ixmp* modelling-platform (adapted from [8]).

This model input-data-handling approach, as such, is novel as it is the first to combine reduced form MS Excel spreadsheet data and lucidly change-tackable .yaml files for input-data documentation. By following the FAIR principles of scientific data-handling and analysis, *d2ix* makes data findable, accessible, interoperable and reusable, and thus facilitates collaborative working and can therefore support the energy-modelling community [9]. By enabling new users to quickly become acquainted with existing models, and by simplifying the generation of new scenarios, *d2ix* reduces the barriers to entry into energy- and climate-policy modelling. Furthermore, the synoptic organisation of the input-data set can reduce the risk of errors prone to happen when organising big data sets (Figure 3). Such errors can have detrimental effects such as the data and coding mistakes causing the infamous Reinhart-Roghoff spreadsheet error [10]. Lastly, the interface will be equipped with a unit test that can inspect the model for commodity 'dead ends' and overly restrictive bounds, a feature that can prevent infeasibilities, undesired exceedingly restricted scenarios and the misinterpretation of results. Overall, *d2ix* is a well-suited data-handling tool for large energy-system models such as MESSAGE$_{ix}$. The easy change-trackable framework for transparent model input-data preparation is the first of its kind to be introduced as a standardised model-creation workflow.



	costs					input		primary output			availability				emissions		initial bounds				growth bounds			
technology	inv_cost	fix_cost	var_cost	technical_lifetime	construction_time	commodity_in1	level_in1	efficiency_1	commodity_out1	level_out1	first_year	last_year	capacity_factor	operation_factor	emission_factor_CO2	emission_factor_CH4	initial_activity_up	initial_activity_lo	initial_new_capacity_up	initial_new_capacity_lo	growth_activity_up	growth_activity_lo	growth_new_capacity_up	growth_new_capacity_lo
coal_ppl	500	30	30	20	1			1,0	electricity	secondary	690	720		1	100						0,1			
wind_ppl	1500	10		20	1			1,0	electricity	secondary	690	720		1							0,1			
grid	300		50	20	1	electricity	secondary	1,0	electricity	final	690	720		1										
bulb	5			1	1	electricity	final	1,0	light	useful	690	720		1										

Figure 3. Reduced spreadsheet input for technology specification for the tutorial.

2. Related Work

While, in the light of good scientific practice, model transparency and reproducibility have received wide academic attention, the focus has remained on how to deal with and how to publish raw-data and model code [11]. However, the important link between the two much noted components—the raw-data and the model—the input-data-handling, has so far not been dealt with scientifically [12,13]. In contrast, the major strategies of input-data-handling which established themselves as go-to solutions in energy-system modelling have never been subject to publication but rather research-institution internal, customised, single-user solutions. Thus, most models now provide different data-handling strategies. Four mayor types can be identified among the most commonly used input-data-handling methods. They are:

- **Type I**—Reduced text-file structures: The input-data of such models is handled in several long or one single, even longer, structured text-file. Such text-file-based input-data systems used to find application with most energy-system models. Due to their long history as well as their suitability for synoptic change-tracking, some modellers still rely on Type I input-data-handling strategies (e.g., *Calliope* [14]).
- **Type II**—Full parameter text-files: The input-data handled by this type is organised in a multitude of text-files. Each file contains one parameter in full dimension and shape as required by the database. Despite the low lucidity and the difficulty in tracking any (un-)intentional changes made to the input-data, Type II data-handling schemes are commonly used. Especially community-friendly, open-source models such as *PyPSA* [15] and *oemof* [16] in particular

appreciate the high flexibility of the input-data-handler in combination with the low requirements regarding the programming skills of the modeller.

- **Type III**—Reduced parameter spreadsheets: Here the input-data is organised in MS Excel spreadsheets in reduced dimension. While this dimension reduction increases the lucidity of the input-data, it can at the same time limit flexibility. However, in order to lower barriers to entry for new modellers, several open-source models such as *urbs* [17] and *ficus* [18] rely on Type III input-data structures.
- **Type IV**—Code-based input-data: Code-based input-data can be either hard-coded or predefined and processed in functions. Thus, the input-data is documented and stored together with the code. Such transparent data-handling types allow for the full documentation of code and input-data within one workflow. However, extensive amounts of hard-coded data, can become overwhelming for any new user, just like the text-file-based data. Nevertheless, several renowned models such as MESSAGE$_{ix}$ [8], *Temoa* [19] and *OSEMOSYS* [20] provide interfaces for hard-coded model input-data.

Table 1 summarises and lists the strengths and shortcomings of those four strategies and compares them to the newly developed *d2ix* workflow. It shows that by filling the gaps in documentation, standardisation and transparency, frameworks such as *d2ix* can help improve energy-system modelling by combining the strength of binary and non-binary input-data storage and handling formats.

Table 1. Comparison of the input-data-handling types used so far and the new data-handling framework *d2ix*. The types are described in Section 2.

	Type I	**Type II**	**Type III**	**Type IV**	**d2ix**
(Git) change-tracking	yes	no	no	yes	yes
Synoptic data presentation	no	no	yes	no	yes
Parameter dimension reduction	no	no	yes	partly	yes
Sub-horizon parameter adaptation	yes	yes	no	yes	partly
Usable without programming knowledge	no	yes	few	no	yes
Dynamic scenario documentation	no	no	no	no	yes
Easy result visualisation	no	no	no	no	yes

3. Methodology

The Python-package we have created, *d2ix*, supports the user in creating new MESSAGE$_{ix}$ models as well as adapting and analysing existing input-data sets and scenarios. The support consists of four main tasks: first, *d2ix* supports the user in organising the input-data for MESSAGE$_{ix}$. For this task, we created an abstracted data model, summarising the reduced model input-data in two spreadsheet files. Secondly, *d2ix* functions as a standardised interface between the spreadsheets and the MESSAGE$_{ix}$ Python API. Third, *d2ix* documents the pre-processed model input-data in yaml text-files. This allows systematic and visual change-tracking of the spreadsheets-based scenario-data using automated change-tracking services such as Git. Lastly, several unit tests implemented in *d2ix* will allow an automated structured inspection of input-data sets to identify commodity 'dead ends' and overly restrictive constraints.

3.1. Class Structure and Definition

The *d2ix* package supports researchers who want to create a MESSAGE$_{ix}$ model, either from scratch or by modifying existing models (Figure 4). This support is supplied by the means of four different classes which handle the data input. In the following, the classes are described in their functionality and structure.

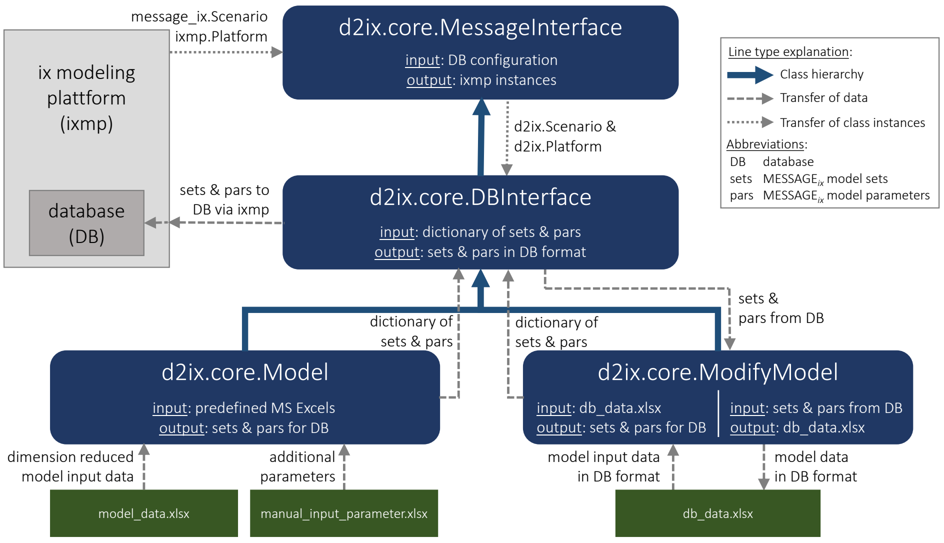

message_ix.Scenario
ixmp.Platform
ix modeling plattform (ixmp)
database (DB)
d2ix.core.MessageInterface
input: DB configuration
output: ixmp instances
d2ix.Scenario & d2ix.Platform
sets & pars to DB via ixmp
d2ix.core.DBInterface
input: dictionary of sets & pars
output: sets & pars in DB format
Line type explanation:
Class hierarchy
Transfer of data
Transfer of class instances
Abbreviations:
DB database
sets MESSAGE_ix model sets
pars MESSAGE_ix model parameters
sets & pars from DB
dictionary of sets & pars
dictionary of sets & pars
d2ix.core.Model
input: predefined MS Excels
output: sets & pars for DB
d2ix.core.ModifyModel
input: db_data.xlsx
output: sets & pars for DB
input: sets & pars from DB
output: db_data.xlsx
dimension reduced model input data
additional parameters
model input data in DB format
model data in DB format
model_data.xlsx
manual_input_parameter.xlsx
db_data.xlsx

Figure 4. Class hierarchy diagram of the *d2ix* package.

3.1.1. MessageInterface—Communication with the Ix Modelling-Platform

The `MessageInterface` class acts as the interface between *d2ix* and the ix modelling-platform (ixmp). To communicate with the ix modelling-platform, `MessageInterface` applies the MESSAGE$_{ix}$ classes `ixmp.Platform` and `message_ix.Scenario`. While the `Platform` instance contains the connection to the database, the `Scenario` class predefines the format and indexation of the model in- and output-data (parameters, sets and variables) required for running the MESSAGE$_{ix}$ model. The database with which `MessageInterface` establishes a communication with is defined in the run-config file provided in the config folder (..\d2ix\config\run_config.yaml.template). The unique identification of the established `Scenario` instance is defined by the user input (Section 4.2), as is the logger setting of the *d2ix* module.

3.1.2. DBInterface—Data-Handling in *d2ix*

The `DBInterface` class enables data-handling in the *d2ix* package. The `DBInterface` class holds the model input-data in the form of a dictionary containing all model sets and parameters which can be accessed and modified before being transferred to the database via the `MessageInterface` class. The central tasks of this class are (i) to hand over the final input-data created in *d2ix* to the database, (ii) to write the final input-data into text-files for transparency and change-tracking, and (iii) to collect the model results from the database after a model run. Furthermore, the `DBInterface` class will check whether the units used in the input-data are already stored in the database and will add them if they are not.

3.1.3. Model—Data Transformation from Reduced Spreadsheet to Database Format

The `Model` class constitutes the core of the *d2ix* package. Its main task is the pre-processing of the input-data from the reduced *d2ix* spreadsheet format to the expanded final input-data format required by MESSAGE$_{ix}$. Apart from creating all required sets and parameters, the `Model` class automatically adds one slack technology for each demand provided in the input-data set, in order to prevent the model from running into infeasibilities during calibration, and to simplify debugging. After each successful scenario-run in MESSAGE$_{ix}$, the `Model` class reformats the results from database tables into time-series elements optimised for post-processing, applying the `TimeSeries` class from the ixmp package [8].

3.1.4. ModifyModel—From Database to Spreadsheet and Back

The `ModifyModel` class is used to enable the analysis and modification of existing MESSAGE$_{ix}$ models, i.e., models readily available in the database. To do so, the `ModifyModel` class has two main functions: (a) `ModifyModel` allows users to choose a specific MESSAGE$_{ix}$ scenario-run, which is then, first collected from the database, secondly, written to an excel sheet and lastly, made accessible to the user as a Python dictionary. The data can then be analysed and modified either in spreadsheet or through scientific computing (e.g., Python). In the second function (b) the modified data can be returned to the database as a new scenario containing the changes applied by the user.

3.2. Testing and User Experience

In accordance with best-practice collaborative programming [21], we set up a Continuous Integration implementation, with CircleCI and Docker each executing several tasks. Additionally two linters, thus static code analysis segments, are configured for basic code quality checks to ensure long term code maintainability. The coding style is tested with Flake8 and MyPy, the static types in Python. Furthermore the API functionality is tested in a defined environment inside a Docker container using the *d2ix* tutorial and some basic examples.

We tested the functionality of *d2ix* together with various beta users. In a first step, the data transfer from the spreadsheet to the *ixmp* platform and the git-tracked text-files was evaluated, thus proving the data-model functionality of *d2ix*. In a second step, we created three models of different sizes, in order to analyse and improve the runtime performance. The model descriptions and runtime performance are documented in Table 2. Finally, we tested the tool's intuitiveness with users without programming skills. By having such a user without programming experience recreating an existing MESSAGE$_{ix}$ model we succeeded in proving the data-model functionally as well as the coherence of the API. As a test model to recreate, we used the standalone country model of South Africa, which is available under the GNU General Public License, Version 3 on GitHub (https://github.com/tum-ewk/message_ix_south_africa) [22]. Two further MESSAGE$_{ix}$ country models are currently being developed for energy-research purposes.

Table 2. d2ix model-creation performance tests with different models. Calculations were performed on a Intel(R) Core(TM) i7 CPU with 3.2 GHz and 64 GB RAM.

	Nodes	Technologies	Historical Periods	Model Periods	Runtime in sec. *
Test Model 1	2	51	5	8	171
Test Model 2	4	88	65	8	657
Test Model 3	16	698	5	8	2291

* average of 10 runs.

4. Tutorial

4.1. Installation

To start using the open source Python-package *d2ix*, you must to ensure that your environment is equipped with the requirements as described in the README instructions found alongside the `d2ix` repository (https://github.com/tum-ewk/d2ix).

4.2. Running d2ix—Creating a Model from Scratch

The core functionality of the *d2ix* tool is to create a model from scratch. The bases for model creation are two reduced spreadsheets (Figure 3). In this example, we create a new MESSAGE$_{ix}$ scenario—in this case the replica of the 'Westeros' tutorial from the MESSAGE$_{ix}$ repository—using the *d2ix* MS Excel templates. The required parameters, configurations and files with the corresponding path are shown in Listing 1. The code creating the scenario is shown in Listing 2 and is explained below.

Furthermore, an introductory tutorial is provided in the *d2ix* repository under `tutorial.ipynb`.

Listing 1: Defining the *d2ix* model-creation parameters.

```
CONFIG = 'config/run_config.yaml'
BASE_XLS = 'input/modell_data_westeros.xlsx'
MANUAL_PARAMETER_XLS = 'input/manual_input_parameter_westeros.xlsx'
MODEL = 'MESSAGE_Westeros'
```

4.2.1. Creating a Model Instance

The `Model` class provides the functionality to create a model from scratch. The class instance is specified by thirteen parameters which are described in Table 3. Furthermore, the code to create a new instance is provided in Listing 2.

Table 3. Parameters used for creating a model instance.

Parameter	**Description**
run_config	the path to the run_config.yaml file located in the config folder [1] *The file contains the specifications of the database type.*
base_xls	the path to the *model_data* file, located in the input folder [2] *MS Excel file consists of seven input sheets that contain all necessary information for the model creation. Thus, the demand, the units, the technologies, and the nodes are defined and mapped in model_data. The structure of the file must to remain unchanged, as the input-data expansion done by d2ix depends on the current structure.*
manual_input_parameter	all possible parameters, such as economic parameters and ecological constraints, can be added to the model using the *manual_input_data* file *It provides the option of adding parameters manually by adding a sheet by the name of the parameter which contains the data in the format required by* $MESSAGE_{ix}$*. Thus, parameters in the manual_input_data are not manipulated by d2ix, but simply scanned for new set elements before being passed on to the database.*
$model_{ixmp}$	the name assigned to the model used as an identifier in the ixmp database
$scen_{ixmp}$	the name assigned to the scenario used as an identifier in the ixmp database
historical_data	allows the use of historical data
first_historical_year	defines the first historical year
first_model_year	defines the first model year If not assigned, it is set to the first year of demand.
last_model_year	defines the last model year If not assigned, it is set to the last year of demand.
historical_range_year	defines the the temporal resolution from historical data
model_range_year	defines the the model temporal resolution
verbose	defines the logger level in order to facilitate easy debugging
yaml_export	allows the export of yaml files from the model *The model parameters and sets can be written to structured text-files before being added to the database in order to facilitate change-tracking (e.g., Git) despite the input-data being provided in xlsx format. This can be turned off during model calibration in order to increase speed.*

[1] ..\d2ix\config\run_config.yaml.template; [2] ..\d2ix\input\model_data.xlsx.

In the example shown in Listing 2, we create an instance of the dummy model 'MESSAGE Westeros', which comes as a tutorial in the *d2ix* repository. The run configurations required for scenario creation with *d2ix* as well as the model input-data paths and the model name are defined in Listing 1. The newly created instance is named 'baseline' and spans over a time horizon from the year 690 to the year of 720. The first model year is defined as the year 700. The resulting model-year vector is equal to [690, 700, 710, 720], wherein 690 is a historical year, thus, not considered in the optimisation.

By setting verbose to true, the log-level is set to debug mode which allows for more information to pass from the creation process to the user. Setting the yaml export parameter to true permits the creation of git-trackable yaml files of the input-data. It is recommended to only set it to false during calibration, as this shortens the model creation runtime, though it disables the git-trackability of the input-data set.

Listing 2: Creating a new MESSAGE$_{ix}$ scenario using the *d2ix* spreadsheet templates.

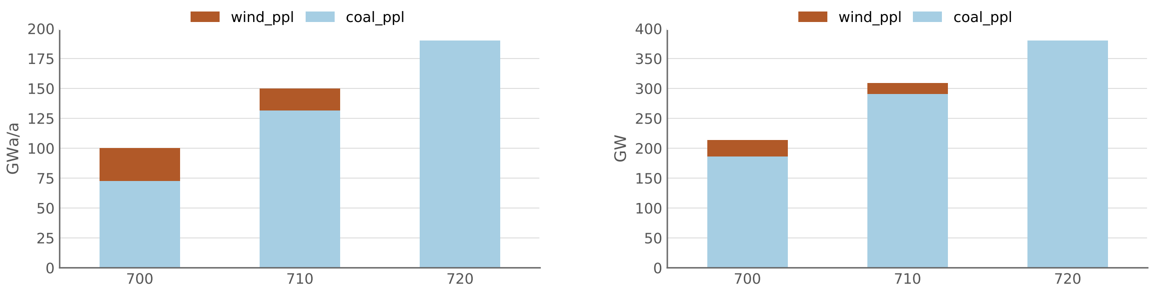
```
1  from d2ix import Model
2
3  # Create a Model instance from the data provided in base_ & manual_parameter_xls
4  d2ix_model = Model(run_config=CONFIG, base_xls=BASE_XLS,
5  manual_parameter_xls=MANUAL_PARAMETER_XLS, model=MODEL, scen='baseline',
6  historical_data=True, first_historical_year=690, first_model_year=700,
7  last_model_year=720, historical_range_year=10, model_range_year=10,
8  verbose=True, yaml_export=True)
9
10 # write data from 'model' dictionary to the database and solve
11 scenario = d2ix_model.model2db()
12 scenario.solve(model='MESSAGE')
13 d2ix_model.close_db()
```

4.2.2. Transferring a Scenario from *d2ix* to the Database—*model2db()*

When the input-data is ready, it can be passed to the database using the *model2db* function, which returns an instance of the `messageix.Scenario` class (Listing 2, line 11).

4.2.3. Solving a Scenario

Using the solve function (from the `messageix.Scenario` class), the database model is dropped to a structured input-gdx file, which is passed on via a solve command to the mathematical model formulation of MESSAGE$_{ix}$. After the successful model run, an output-gdx file is created containing all input and output-data. This file content is automatically passed on to and stored in the database. Further details on the *solve()* function can be found in the MESSAGE$_{ix}$documentation [23]. Sample results of the baseline scenario from the Westeros example are shown in Figure 5.



(**a**) Activity (**b**) Installed capacity

Figure 5. Power plant activity and capacity in the 'baseline' scenario (Listing 2).

4.2.4. Modifying the Input-Data—*get_parameter()*, *set_parameter()*

After creating the class instance, `model` contains a dictionary of all parameters and sets of the expanded input-data, which can now be accessed (Listing 3, line 11), modified (line 12) and returned to the dictionary (line 13). In this scenario we introduce an emission tax using the *d2ix get_parameter*, *set_parameter* procedure. The comparison between Figures 5 and 6 visualises the change in results induced by the introduction of the tax.

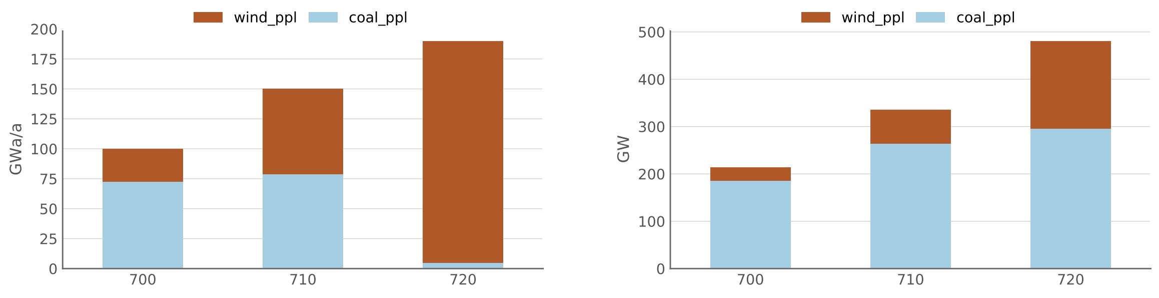
Listing 3: Creating a MESSAGE$_{ix}$scenario—with a carbon tax—using the *d2ix get_parameter()* and *set_parameter()* approach.

```python
from d2ix import Model

# Create a Model instance from the data provided in base_ & manual_parameter_xls
d2ix_model = Model(run_config=CONFIG, base_xls=BASE_XLS,
manual_parameter_xls=MANUAL_PARAMETER_XLS, model=MODEL, scen='tax-emission',
historical_data=True, first_historical_year=690, first_model_year=700,
last_model_year=720, historical_range_year=10, model_range_year=10,
verbose=True, yaml_export=True)

# Add a emission tax
tax_emission = d2ix_model.get_parameter(par='tax_emission')
tax_emission['value'] = [0.264, 0.429, 0.699]
d2ix_model.set_parameter(par=tax_emission, name='tax_emission')

# write data from 'model' dictionary to the database and solve
scenario = d2ix_model.model2db()
scenario.solve(model='MESSAGE')
d2ix_model.close_db()
```



(**a**) Activity (**b**) Installed capacity

Figure 6. Power plant activity and capacity in the 'tax-emission' scenario (Listing 4).

4.3. *Running d2ix— Modifying Existing Models*

The *d2ix* package can also be used to modify existing models. The code required for retrieving, modifying and returning input-data sets to the database is shown in Listing 4, and is explained below.

4.3.1. Creating a ModifyModel Instance

The `ModifyModel` class provides the functionality of collecting models from the database, writing them into a structured spreadsheet file for user modification and returning the modified model to the database. The parameters specifying the `ModifyModel` instance not introduced in Section 4.2.1 (run_config, model, scen and verbose) are described in Table 4.

Table 4. Parameters used for creating a modified model instance.

Parameter	Description
xls_dir	the path to the directory where the MS Excel file will be saved
file_name	the name of the MS Excel that will be created or that the data will be read from

4.3.2. From Database to ModifyModel Instance & Excel Sheet—*scen2xls()*

The *scen2xls()* function (Listing 4, line 8) searches the database for the scenario defined by model and scenario name, and in the `mod_model`. If a scenario with the defined model and scenario name is available, all parameters and sets from the most recent (default) version of the scenario will be written to the spreadsheet. If a version is specified, this version instance of the scenario will be copied.

4.3.3. From the Excel File to ModifyModel Instance—*xls2model()*

The *xls2model()* (Listing 4, line 10) function reads the spreadsheet file specified in the `mod_model` instance and stores the data as a structured dictionary in the instance. The data is then available to modify, analyses and visualise using the Python functionality.

Listing 4: Modifying an existing MESSAGE$_{ix}$ model using spreadsheet inputs.

```
1  from d2ix import ModifyModel
2
3  # Create a ModifyModel instance
4  mod_model = ModifyModel(run_config=CONFIG, model=MODEL, scen=SCEN, xls_dir='xls_folder',
5  file_name='db_data.xlsx', verbose=False)
6
7  # Collecting a scenario from the database and saving it to an MS Excel file
8  mod_model.scen2xls(version=None)
9  # Collection a scenario from a MS Excel file and saving it to the database
10 mod_model.xls2model()
11
12 # write data from 'mod_model' dictionary to the database and solve
13 scenario = mod_model.model2db()
14 scenario.solve(model='MESSAGE')
15 mod_model.close_db()
```

4.4. *Post-Processing a MESSAGEix Scenario*

The *ixmp* package supplies tools for standardised reporting of reference data and results. These tools are documented and described in [8] as well as in the online documentation [24].

5. Conclusions

In *d2ix*, we built a package that supports users in creating, modifying, and analysing MESSAGE$_{ix}$ scenarios. The main benefits of using *d2ix* for scenario creation are threefold. (i) The synoptic input-data supports the transparency and reproducibility of even large models and can thus reduce errors. It further encourages collaborative modelling attempts by making it easier to understand and review model parameters and assumptions implemented by other researchers. (ii) By reducing the dimensions of the input-data, the researchers can easily handle the data using two MS Excel sheets. Hence, *d2ix* reduces barriers to access by reducing input-data complexity and allowing scenario creation without programming knowledge. (iii) *d2ix* permits the combination of the benefits of 'higher' (easy and synoptic data-handling) and 'lower' (change-trackability) data formats. To put it succinctly: by providing a synoptic and easy input-data-handling workflow *d2ix* can support the efforts of the open data movement within the MESSAGE$_{ix}$ modeller community and can serve as an example for data-handling frameworks built for other model types.

However, simplification of input-data does reduce the flexibility of the model, e.g., currently a maximum of two outputs is supplied for each technology. However, this can be bypassed by either adapting the model parameter 'output' using the *get_* and *set_ parameter* functionality, or by adapting the input spreadsheet and the underlying code to supply as many outputs as required. An expansion of *d2ix* to increased flexibility could be subject of future work; however, the decision on the specific balance between flexibility and simplicity requires practical experience which still remains to be collected.

Author Contributions: T.Z. and C.L.O. have cooperated in the development of *d2ix*. While T.Z. is the creator of most of the code and software concept, C.L.O. developed the work flow and the overlying package concept, which are described in this paper. Both authors read and approved the final manuscript.

Funding: This work was supported by the German Research Foundation (DFG) and the Technical University of Munich (TUM) in the framework of the Open Access Publishing Program.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Schrattenholzer, L. *The Energy Supply Model Message*; Number 81-31 in Research Report; OCLC: 254145200; International Institute for Applied Systems Analysis (IIASA): Laxenburg, Austria, 1981.
2. Messner, S.; Schrattenholzer, L. MESSAGE–MACRO: Linking an energy supply model with a macroeconomic module and solving it iteratively. *Energy* **2000**, *25*, 267–282. [CrossRef]
3. Koomey, J.; Berard, S.; Sanchez, M.; Wong, H. Implications of Historical Trends in the Electrical Efficiency of Computing. *IEEE Ann. Hist. Comput.* **2011**, *33*, 46–54. [CrossRef]
4. Messner, S.; Strubegger, M. The energy model MESSAGE III. In *Advances in Systems Analysis: Modelling Energy-Related Emissions on a National and Global Scale*; Hake, J.F., Kleemann, M., Kuckshinrichs, W., Martinsen, D., Walbeck, M., Eds.; Konferenzen des Forschungszentrums Juelich: Juelich; Germany; 1994.
5. Huppmann, D.; Rogelj, J.; Kriegler, E.; Krey, V.; Riahi, K. A new scenario resource for integrated 1.5 °C research. *Nat. Clim. Chang.* **2018**, *8*, 1027–1030. [CrossRef]
6. Fricko, O.; Havlik, P.; Rogelj, J.; Klimont, Z.; Gusti, M.; Johnson, N.; Kolp, P.; Strubegger, M.; Valin, H.; Amann, M.; et al. The marker quantification of the Shared Socioeconomic Pathway 2: A middle-of-the-road scenario for the 21st century. *Glob. Environ. Chang.* **2017**, *42*, 251–267. [CrossRef]
7. Baker, T.; Asim, M.; Tawfik, H.; Aldawsari, B.; Buyya, R. An energy-aware service composition algorithm for multiple cloud-based IoT applications. *J. Netw. Comput. Appl.* **2017**, *89*, 96–108. [CrossRef]
8. Huppmann, D.; Gidden, M.; Fricko, O.; Kolp, P.; Orthofer, C.; Pimmer, M.; Kushin, N.; Vinca, A.; Mastrucci, A.; Riahi, K.; et al. The MESSAGE 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. *Environ. Model. Softw.* **2019**, *112*, 143–156. [CrossRef]
9. Wilkinson, M.D.; Dumontier, M.; Aalbersberg, I.J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.W.; da Silva Santos, L.B.; Bourne, P.E.; et al. The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* **2016**, *3*, 160018. [CrossRef] [PubMed]
10. Reinhart, C.M.; Rogoff, K.S. Growth in a Time of Deb—Errata. *Am. Econ. Rev.* **2010**, *100*, 573–578. [CrossRef]
11. Pfenninger, S. Energy scientists must show their workings. *Nature* **2017**, *542*, 393–393. [CrossRef] [PubMed]
12. Pfenninger, S.; Hirth, L.; Schlecht, I.; Schmid, E.; Wiese, F.; Brown, T.; Davis, C.; Gidden, M.; Heinrichs, H.; Heuberger, C.; et al. Opening the black box of energy modelling: Strategies and lessons learned. *Energy Strategy Rev.* **2018**, *19*, 63–71. [CrossRef]
13. Cao, K.K.; Cebulla, F.; Vilchez, J.J.G.; Mousavi, B.; Prehofer, S. Raising awareness in model-based energy scenario studies—Transparency checklist. *Energy Sustain. Soc.* **2016**, *6*. [CrossRef]
14. Pfenninger, S.; Pickering, B. Calliope: A multi-scale energy systems modelling framework. *J. Open Source Softw.* **2018**, *3*, 825. [CrossRef]
15. Brown, T.; Hörsch, J.; Schlachtberger, D. PyPSA: Python for Power System Analysis. *J. Open Res. Softw.* **2018**, *6*. [CrossRef]
16. 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 Rev.* **2018**, *22*, 16–25. [CrossRef]
17. Dorfner, J. Open Source Modelling and Optimisation of Energy Infrastructure at Urban Scale. Ph.D. Thesis, Technical University of Munich, Munich, Germany, 2016.
18. Atabay, D. An open-source model for optimal design and operation of industrial energy systems. *Energy* **2017**, *121*, 803–821. [CrossRef]
19. Decarolis, K.H.S.S.J.F. Modeling for insight using Tools for Energy Model Optimization and Analysis (Temoa). *Energy Econ.* **2013**, 339–349. [CrossRef]
20. Howells, M.; Rogner, H.; Strachan, N.; Heaps, C.; Huntington, H.; Kypreos, S.; Hughes, A.; Silveira, S.; DeCarolis, J.; Bazillian, M.; et al. OSeMOSYS: The Open Source Energy Modeling System. *Energy Policy* **2011**, *39*, 5850–5870. [CrossRef]

21. Latte, B.; Henning, S.; Wojcieszak, M. Clean code: On the use of practices and tools to produce maintainable code for long-living. In Proceedings of the Workshops of the Software Engineering Conference 2019, Stuttgart, Germany, 18 February 2019.
22. Orthofer, C.L.; Huppmann, D.; Krey, V. South Africa After Paris—Fracking Its Way to the NDCs? *Front. Energy Res.* **2019**, *7*, 20. [CrossRef]
23. International Institute for Applied Systems Analysis (IIASA). The MESSAGEix Framework Documentation. 2018. Available online: http://messageix.iiasa.ac.at (accessed on 8 April 2019).
24. International Institute for Applied Systems Analysis (IIASA). The MESSAGEix Framework. 2018. Available online: https://github.com/iiasa/message_ix (accessed on 8 April 2019).