pytrnsys

Dec 06, 2022

Contents

A short presentation (15 min) of pytrnsys and its features can be found in the following YouTube video.

The pytrnsys package provides a complete python-based framework to build, post-process, plot, and report TRNSYS simulations. It is designed to give users a fast, fully automatized, and reproducible way to execute and share TRNSYS simulations by the use of a single short configuration file. In addition, a large variety of commands is accessible to post-process simulation results in one shot. This functionality extends beyond processing TRNSYS generated data and can also be used for analyzing generic data. These could be, e.g., measurement data that are structured in a similar way as the TRNSYS results.

The package was developed at the SPF - Institute for Solar Technology at the OST - Eastern Switzerland University of Applied Sciences.

CHAPTER 1

Getting started

1.1 Installation

pytrnsys is best installed as part of its graphical user interface (see also below). For a standalone installation of the pytrnsys python package, please follow the installation guide on GitHub.

1.2 Graphical user interface

pytrnsys can be combined with a graphical user interface (GUI). This GUI is a separate package that includes pytrnsys as a requirement. For further information on the GUI and on how to install it, please visit its GitHub page.

1.3 Prerequesites

At the moment, TRNSYS 17 and TRNSYS 18 (32 bit) are supported. In order to use pytrnsys, you need to have TRNSYS installed. Several additional types are delivered with pytrnsys. For those you'll manually need to copy the dll-files from:

```
pytrnsys_data\ddcks\dlls
```

to the respective folder of your TRNSYS installation:

```
...\UserLib\ReleaseDLLs
```

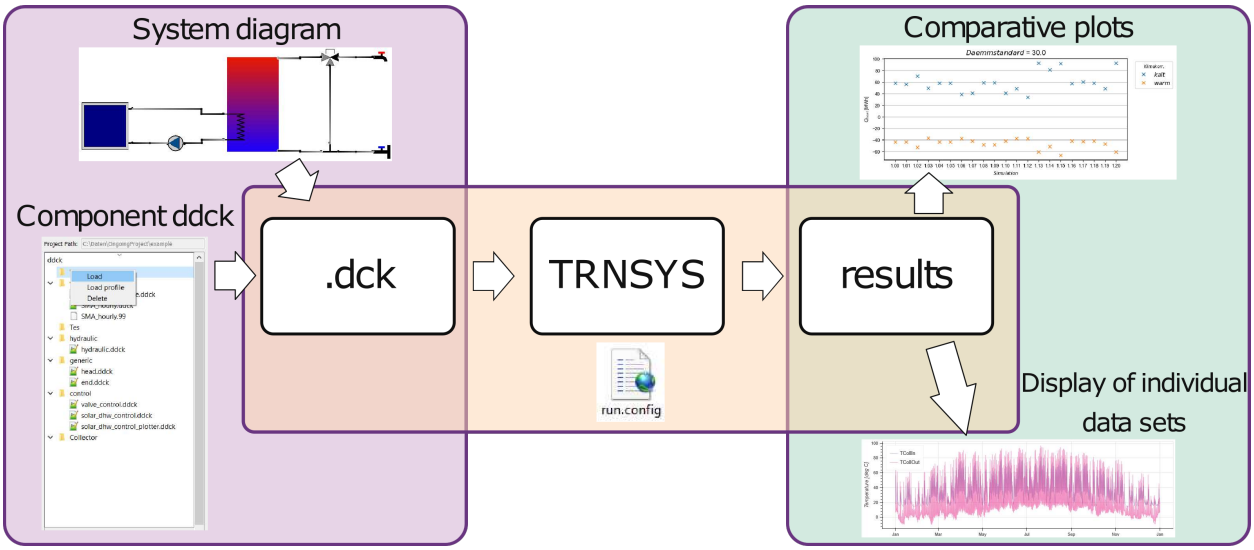
Additional optional prerequisites are:

- A working LaTeX distribution. We recommend MiKTex due to its included package management system.
- Automated plotting is done by matplotlib. GLE is needed to create Q-vs-T plots using the commands *plotHourlyQvsT* or *plotTimestepQvsT*

- GLE is also supported by the configuration file keyword *setPrintDataForGle* which exports a .gle file which can be used for further plotting in GLE.
- Inkscape can be used to save the plots in the enhanced meta file format by using the *plotEmf keyword* in the processing.

1.4 The philosophy of pytrnsys

When using TRNSYS to simulate systems, one arguably goes through three steps, which are all supported by pytrnsys. A graphical user interface and a modular approach facilitate the creation of a dck-file, which defines the simulation. Configuration files containing simple command lines support the user in specifying how they want to run the simulation or do a parametric study. Finally, another type of configuration file allows to automatically do calculations on the simulation results and generate plots.



One a more detailed level this means:

1. **Building a dck-file, which defines the simulation**
    - The concept of pytrnsys is to use a modular approach stacking files with the extension *.ddck together to form a single dck TRNSYS file.
    - The ddck files are structured in a way that they can be reused/modified easily to adpat to new cases. These files should be uploaded to GIT repositories if sharing/reusing is foreseen.
    - Our core idea to build a TRNSYS dck is to use a flow solver and an hydraulic ddck-file which is custom to each case. The TRNSYS flow solver is a TYPE that takes the set mass flow rates of pumps and the positions of 3-way controlled valves as an input for each time step and returns the mass flow rates through all pipes and components.
    - The hydraulic ddck-file also includes all TYPEs for the hydraulic elements such as pipes and tee-pieces. Thus, when connecting to all elements such as solar collectors the mass flow and temperature of the pipe that enters the collector which has a specific format name can be used directly. That is, connection between elements is very easy and can be done in a fully automatic way.
    - One of the functionalities of the GUI is to export the hydraulic setup such that it can be used directly with the flow solver. The hydraulic files you will find in the examples are exported from our GUI.
    - If you don't have the GUI you can still work with pytrnsys without any problem. However, you will need to know/connect the inputs (mass flow rates and temperatures) for each component like in normal

TRNSYS. Our GUI and the flow solver make this almost fully automatic.

2. **Running the simulation(s)**

   - Once a TRNSYS dck has been generated with the method described above, by your own method or by Studio you can execute this dck with a lot of nice functionalities. For example you can easily run parametric studies in parallel and modifiy the dck file using a configuration file.

3. **Post-processing the simulation results**

   - Once the simulations are done you can easily process all results including several results from parametric studies using a config file where the main processing calculations can be done.
   - Some automatic processing is always done. For example the energy demand and the energy balance of the system is calculated automatically provided a proper syntax is used in the TRNSYS dck.
   - The custom-made processing can be easily added. To fully use our processing functionality you need a working LaTeX environment.
   - The processing functionality includes monthly and hourly calculations, files with results, and different types of automatic plots.
   - Basically all functionality can be added such that it can be activated from the config file. For more project specific processing it might be better to work on python level. You will see how to do this at the developer's guide section.
   - Our method of processing TRNSYS simulations is based on our method to build a TRNSYS dck, so to fully use all functionalities you will need to change your own dck to have a similar structure as the one we have. For example the results are always stored in a temp subfolder and to do the automatic energy balance you need to provide the data with specific namings convention. However, still many functionalities can be theoretically done if you don't follow our method and style, but we never checked this, so you might find issues there.

This package is not intended to substitute your skills in TRNSYS, but if you have them it will make your life easier. For those that don't know TRNSYS yet it will make the introduction easier, or at least this is our hope.

CHAPTER 2

Building a dck-file

2.1 Modular ddck-files

Note: This section only provides a short introduction to ddck-files. A more detailed description can be found in *the dedicated section*

One central aspect of pytrnsys is modularization. A TRNSYS simulation is fully defined by an often lengthy dck-file containing all relevant information. Several UNITS of different and/or the same TYPES can be contained in such a file. To make the creation of such a dck-file more flexible pytrnsys composes a dck-file from several modular files, which were given the novel extension `ddck`. Though there are a few special cases such ddck-file are typically corresponding to one component in the system simulation, e.g. a heat pump. The ddck-files are structured in the following manner:

1. Name of the component
2. Change log
3. Description
4. Inputs from hydraulic solver
5. Outputs to hydraulic solver
6. Outputs to energy balance
7. Dependencies with other ddck-files
8. TYPE declaration (PARAMETERS, INPUTS and OUTPUTS)
9. Printer declaration(s)
10. Online plotter declaration(s)

There are a few special ddck-files to which this outline does not apply. Those are described in the following section.

2.2 Composing a dck-file

The modular ddck-files are stacked together to form a dck-file:

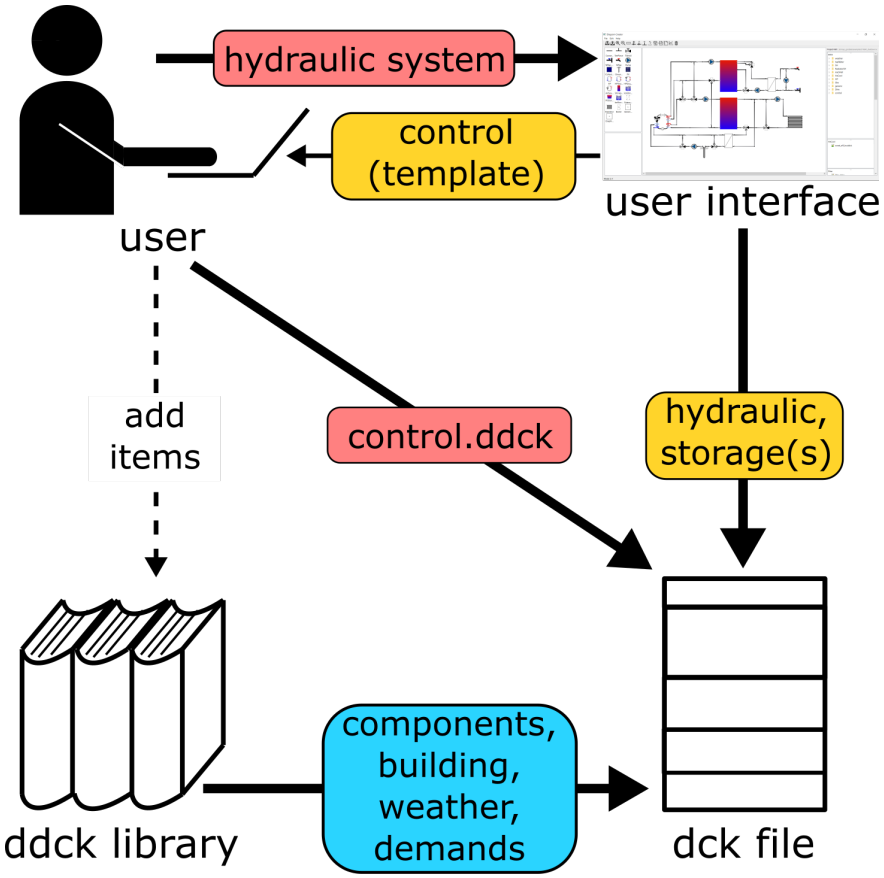
```
head.ddck
component_1.ddck
component_2.ddck
...
component_n.ddck
hydraulic.ddck
control.ddck
end.ddck
```

Here the `head.ddck` contains the VERSION statement and a few global constants and the `end.ddck` simply contains the END statement that each dck-file needs to include. Other special ddck-files that do represent a component of the system to be simulated is the `hydraulic.ddck` and the `control.ddck`. The former is based on type 935, which was developed at SPF and contains the hydraulic information of the system, i.e. which component is connected which other component and which pipes are used for that. The `control.ddck` specifies the position of valves in the system and the mass flow rates of pumps. A template for this file can be exported from the graphical user interface. Note that ddck-files that, e.g., specify the weather data are treated like system components.

Each standard ddck-file is connected to the hydraulics of the system through its inputs from and outputs to the hydraulic solver. The inputs are the mass flow rates and temperatures of the pipes providing the hydraulic inputs to the components, while the outputs are the respective outlet temperatures.

2.3 Workflow

There are three sources for the ddck-files that make up a dck-file. The primary source is the ddck library that is delivered with pytrnsys. It contains a wide array of system components like solar collectors or heat pumps and many more. The second possible source is the user, who can also employ the graphical user interface either as the third source of ddck-files or to generate templates of ddck-files.

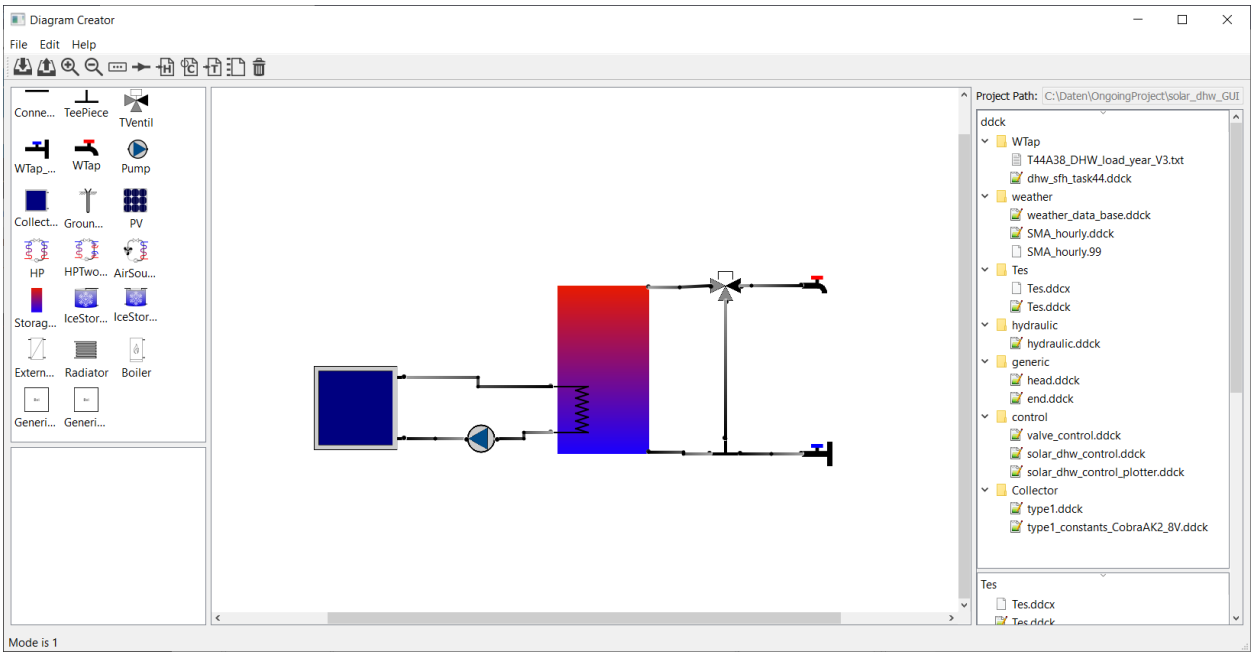


ddck-files that are specific to each system can be directly exported from the graphical user interface. These are the thermal storage tank(s) and the hydraulic information of the system. Furthermore, a template for the `control.ddck` can be exported from the graphical user interface, of which the details need to be filled by the user directly. Finally, a user can also create any kind of ddck-file specific to their needs from scratch.

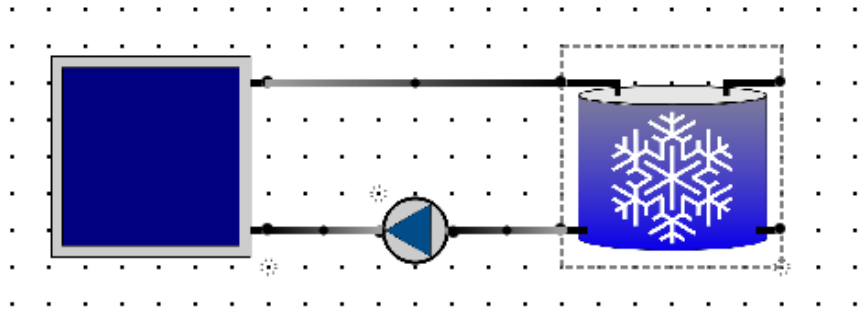
CHAPTER 3

The graphical user interface

The user interface consists of a menu bar, a tool bar, and three widgets. The central and main widget of the interface is the diagram. Here different components can be placed and connected. The available components can be found in the widget on the lefthand side and can simply be dragged into the diagram to be placed. On the righthand widget the file tree of the `ddck`-folder can be seen. For each component dropped into the diagram an additional folder is created in this tree.



Components are simply added to a diagram by dragging the respective item from the library on the left into the central widget. It is recommded to toggle the snap grid for placing components to ease the alignment of them:



Connections can be created by moving the cursor over a port, pressing the mouse, and dragging the cursor to the port that is supposed to be connected, and release the mouse there. Each pipe has a technical flow direction along which the mass flow rate is considered to be positive (and negative if it goes in the opposite direction). The technical flow direction is indicated by a gradient going from a bright to a darker grey along the positive direction.

3.1 The menu and the tool bar

3.1.1 File

`New` Create a new project.

`Open` Open an existing project.

`Save` Save the current diagram.

`Copy to new folder` Copy the complete content of the current project folder to a new one.

`Export as PDF` Export the current diagram as a pdf.

`Debug Conn` TBD

`Set TRNSYS path` Set the path to the exe-file of the TRNSYS simulation from which you want to run the mass flow solver.

3.1.2 Project

`Run mass flow solver` Run a short TRNSYS simulation of the hydraulics of your system with the valve positions and pump mass flow rates as given by the respective dialogues.

`Start mass flow visualizer` Visualize the mass flow rates and temperatures of the pipes of the hydraulic system.

`Export hydraulic.ddck` Export the ddck-file of the hydraulics of your system.

`Export hydraulic_control.ddck` Export a template of the ddck-file specifying the control of your hydraulic system.

`Update run.config` Update the configuration file for the execution of the simulation with the current project path and the ddck-files included into the ddck-folder of your project.

`Export dck` Export a dck file according to your configuration file, but do not launch the simulation.

`Run simulation...` Run the simulation(s) as specified by the run.config-file.

`Process simulation...` Post-process the results of your simulation as specified by the process.config-file.

`Export json-file containing connection information` Export a json-file specifying each port's variable name for mass flow rate and temperature.

3.1.3 Edit

`Toggle snap grid` Toggle a grid to which the components can snap to ease placing them.

`Toggle align mode` TBD

`Undo` Undo the last edit.

`Redo` Redo the last undone edit.

3.1.4 Remaining tool bar items

`Zoom in` Zoom into diagram.

`Zoom out` Zoom out of diagram.

`Toggle labels` Toggle the labels of the components and pipes.

`Delete diagram` Deletes the current diagram.

3.2 Component library and port information

3.2.1 Port information

The following information is shown on the widget on the bottom left of the GUI when the cursor is hovering over a port:

`ID` A unique identifier that is associated with each port when a component is dropped in the diagram.

`Names` The name of the port and its default function (input or output) determining the positive technical flow direction through the component. The port name is used in the ddck-file corresponding to the component.

`Block` The name of the component to which the port is attached.

`Connections` The name of the pipe that is connected to the port. This name will be used when the placeholder statements are automatically replaced.

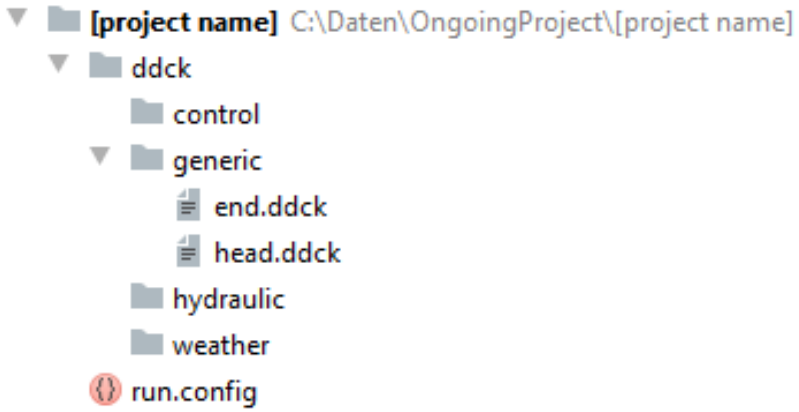
3.3 The file structure of a project

The GUI expects a certain file structure for a directory representing a project. When a new project is initialized this structure is automatically created and it needs to be there, when one wants to open a project with the GUI. In the following the folder will be called `[project name]`, where the square brackets indicate a placeholder for an actual name. When the GUI is started the user is asked whether they want to create a new project or open an exisiting one. When a new project is created a dialogue guides the user through the creation of a new project folder. Saving a diagram is accomplished through the generation of a json-file inside `[project name]`, that has the same name. This means that in order to open the existing `[project name]` the following file needs to be opened:

```
..\[project name]\[project name].json
```

All files that need to be loaded for the project or which are generated from the GUI are saved in `[project name]`.

When a project is initialized the following file structure is created:



The folder called `ddck` contains the folder `generic`, into which the generic ddck-files `head.ddck` and `end.ddck` are loaded. Furthermore, the empty folders `control`, `hydraulic`, and `weather` are created. These are folders for ddck-files that are not directly connected to any components in a project diagram. The project folder also contains `run.config`, which is a template that can be altered by the user.

3.3.1 Handling ddck-files

When dropping a component that is supposed to be represented by a ddck-file on the simualtion level, a folder is created in the ddck-folder of the project that dynamically changes its name with the name of the component. All files that are needed to represent the respective component when building a dck should be loaded into this component folder.

In the following different folders are described, which can be found in:

```
..\[project name]\ddck
```

Square brackets indicate place holders for component names.

`generic`

This folder is created when a project is initialized and holds `head.ddck` and `end.dck`, which contain the information that needs to be added at the beginning and the end of a dck-file respectively.

`[storage tank]`

As soon as a storage tank is dropped in the diagram, a folder of the same name is created. Its name changes dynamically with the name of the storage tank. When the ddck-file of the storage tank is exported, it will build the following two files:

```
..\ddck\[storage tank]\[storage tank].ddck
..\ddck\[storage tank]\[storage tank].ddcx
```

Here the file with the extension `.ddck` contains the information of the storage tank, that is needed to build the dck-file. Meanwhile, the file with the extension `.ddcx` contains the black box component temperature equations, which are needed to build `hydraulic.ddck` (see below).

`[component (not a storage tank)]`

As soon as a component that requires one or more ddck-files for a simulation is dropped in the diagram, a folder of the same name is created. Its name changes dynamically with the name of the component. The user needs to load the ddck-file(s) that represent the component in question into the corresponding folder.

`hydraulic`

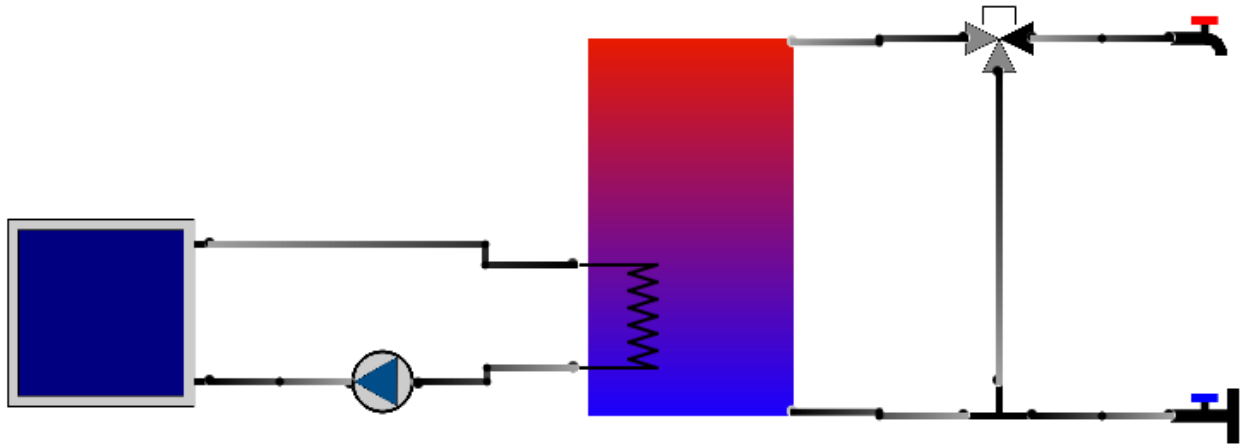
This folder is created when a project is initialized. It is the default export destination for `hydraulic.ddck`.

`control`

This folder is created when a project is initialized. The user should load all ddck-files which represent control features into this folder. It is the default export destination for `hydraulic_control.ddck`.

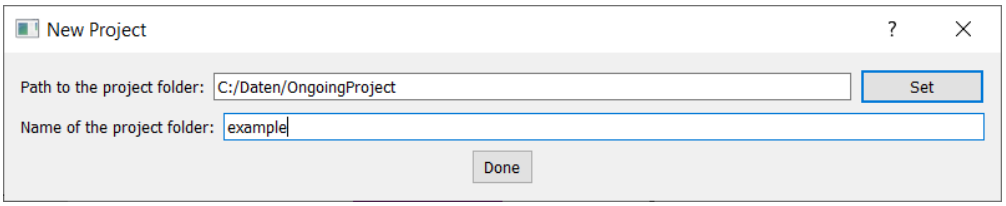
3.4 Complete procedure example

In the following the step by step procedure for starting with an empty diagram to launching a TRNSYS simulation is presented. The full diagram of the demonstrated example looks like:

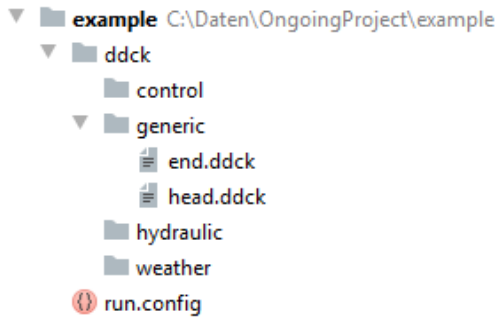


3.4.1 1. Initialize a project

When a new project is initialized the following dialogue is opened to build a folder for the project:



This creates the following file tree:

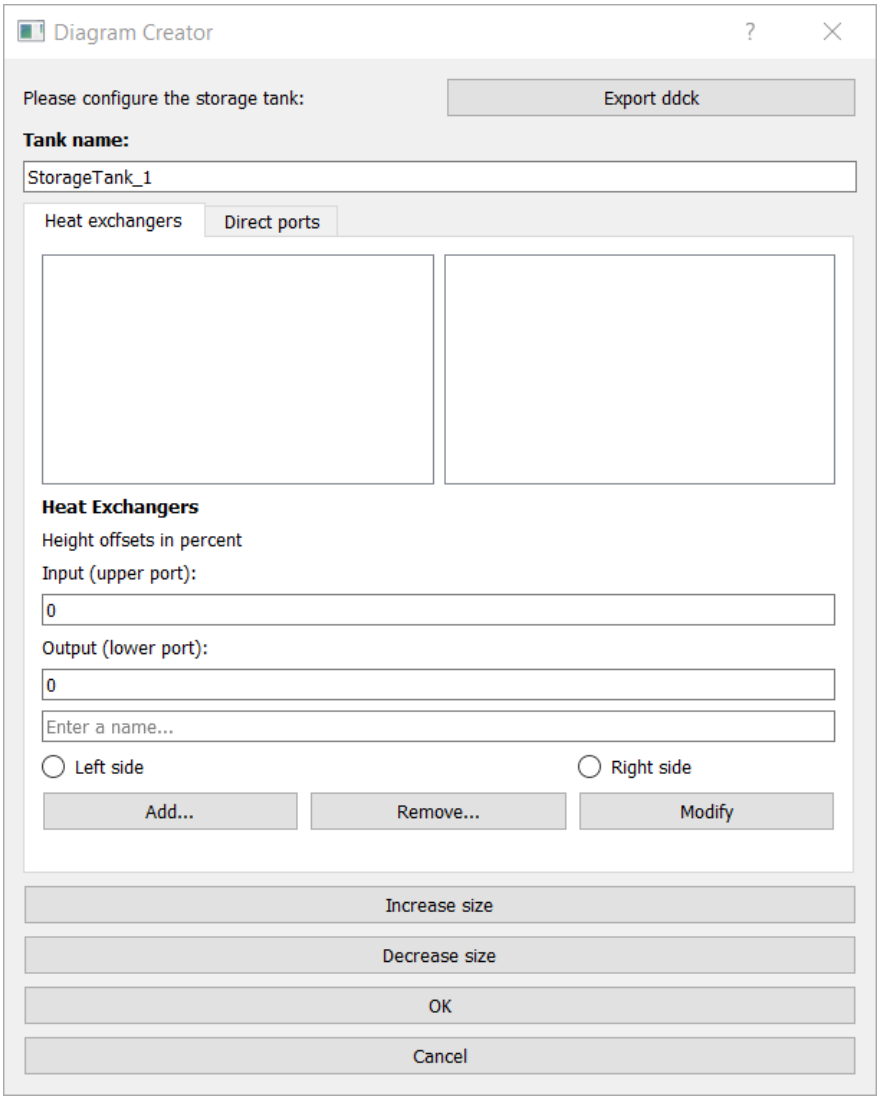


3.4.2 2. Build a diagram

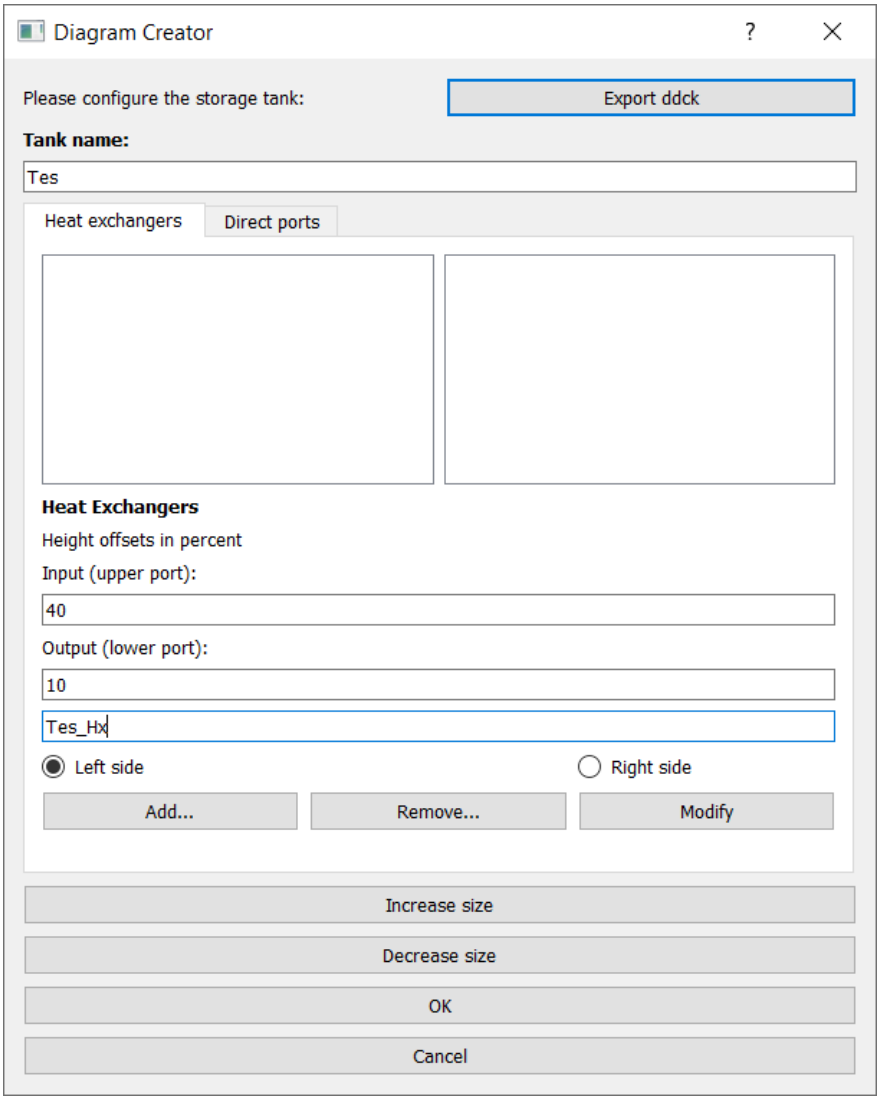
To make the placement of the components easier the snap grid is toggled.

(i) Set up storage tank

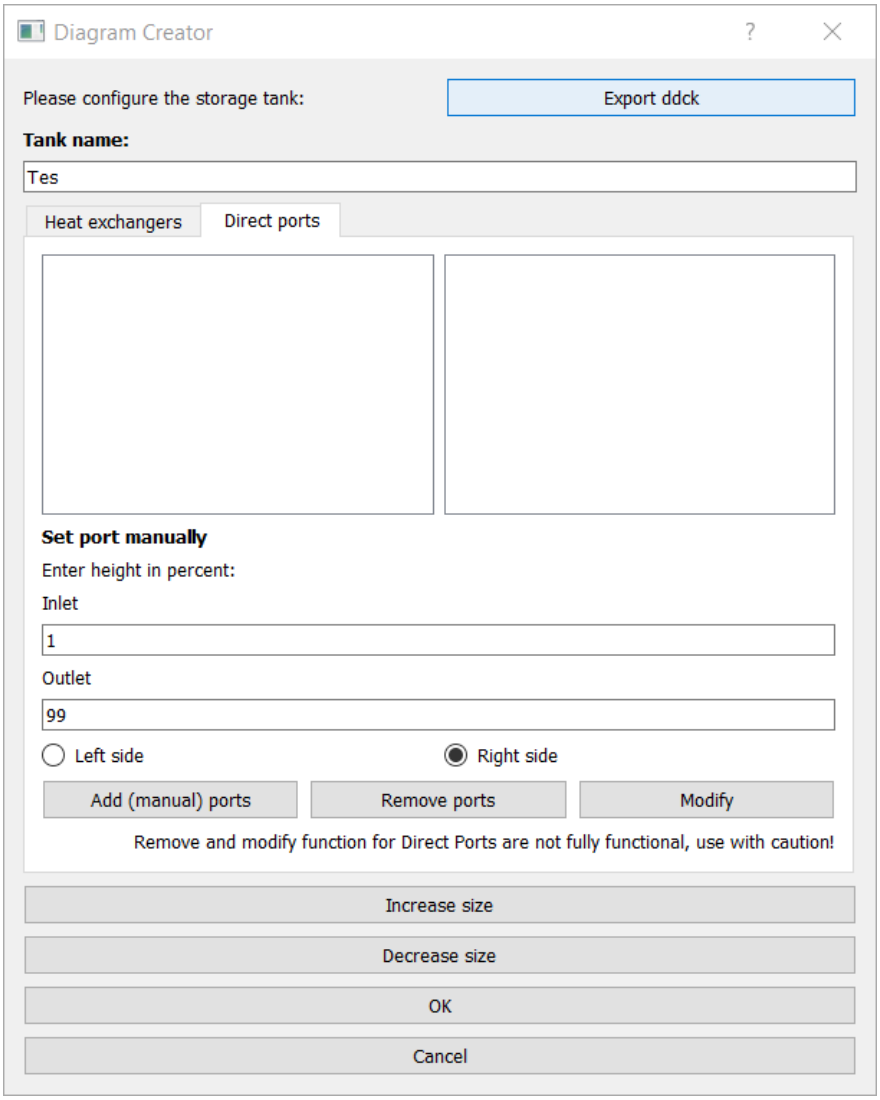
First, a storage tank is dropped in the diagram. This opens the following dialogue:



To make the diagram better arranged, the size of the storage is increased. Furthermore, its name is changed to ‘’Tes”. Then a heat exchanger with its input at 40 % and its output at 10 % height of the storage tank on the left side is added and named ‘’Tes_Hx”:

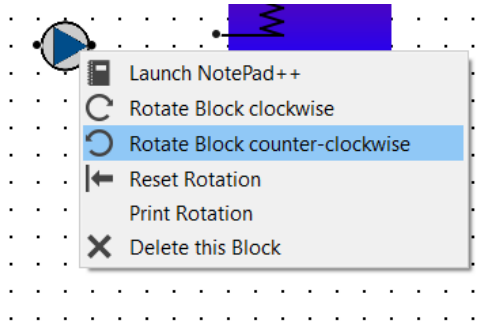


Additionally, a pair of direct ports is added on the right side with the input at 1 % and its output at 99 % height of the storage tank:

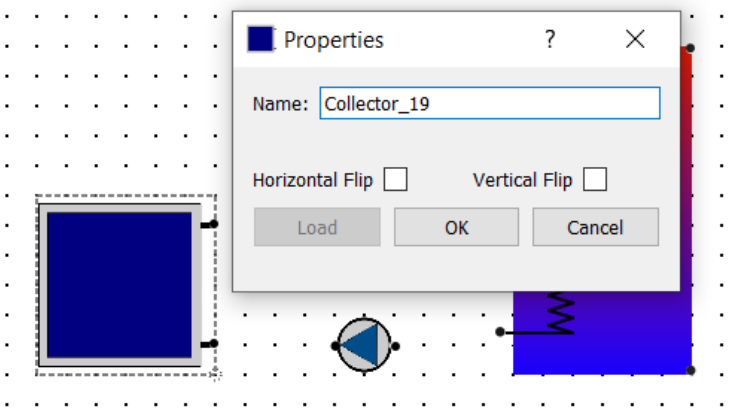


(ii) Place components

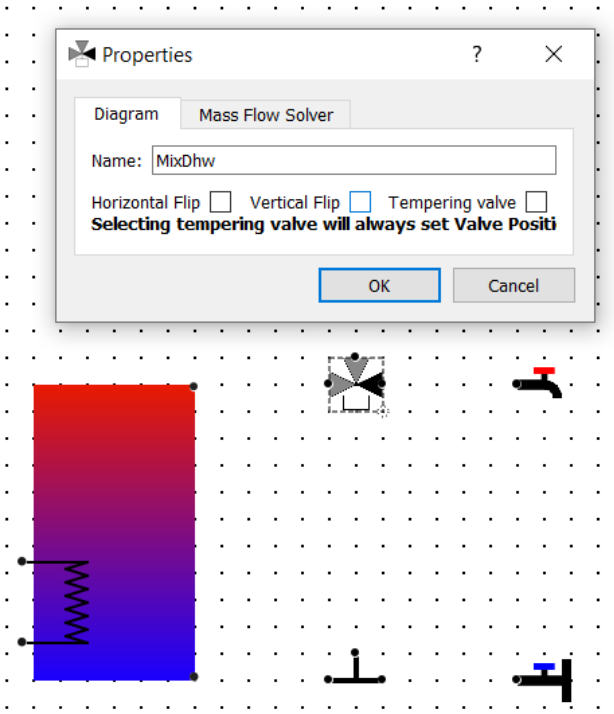
Next, the components are placed one by one. They can be rotated when they are right-clicked, which is needed in the current example for the pump.



Furthermore, when double-clicking components they can be re-named, which will also change the name of their respective folder in the ddck-directory.

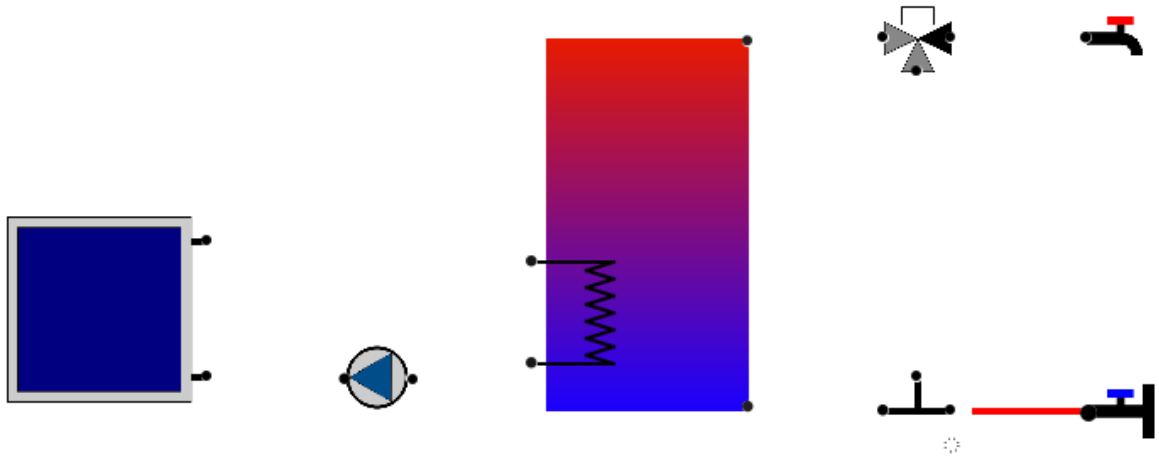


When a three-way valve is placed, the darker connector indicates the port through which there is always flow, when there is flow at all. In our case this needs to be towards the warm water tap, since the three-way valve placed is supposed to be the mixing valve for the warm water demand. To access the valve settings the component is simply double-clicked. In the opened dialogue "vertical flip" is ticked for the correct orientation of the valve and "tempering valve" to make it one.

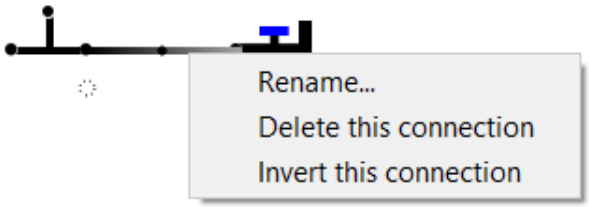


(iii) Connect ports

After placing all components they can be connected. A connection is made from a port by holding the left mouse key and dragging the cursor to another port, where the mouse is released.

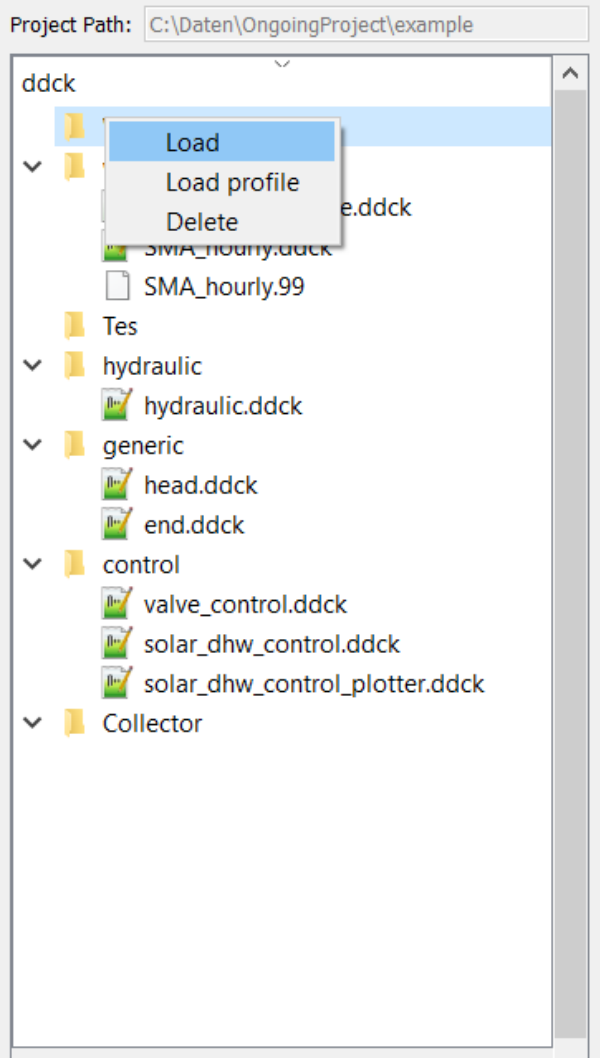


The dialogue for a connection can be opened by right-clicking it.



3.4.3 3. Load ddck files

Many components need to be represented by ddck- or other files in the simulation. These files need to be loaded to the individual component folders in the ddck-directory. This is done by right-clicking the respective folder in the file tree.



3.4.4 4. Export files and launch TRNSYS simulation

Once the component files are loaded, different files needed for the simulation can be exported from the diagram.

(i) Export Tes.ddck

First, the ddck of the thermal storage tank needs to be exported. This is done by opening the storage tank dialogue by double-clicking the component and then hitting the respective button on this dialogue.

(ii) Export hydraulic.ddck

Next, the ddck representing the hydraulics of the system needs to be exported. This is done by hitting the respective button in the tool bar.

(iii) Export DdckPlaceHolderValues.json

After that, the json-file consisting of the variable names for mass flow rate and temperature for each port needs to be exported. This is done by hitting the `Export json-file containing connection information` button under `Project` tool bar. Following shows the structure of a DdckPlaceHolderValues.json:

```
{
    "ComponentName": {
            "InputPortName": {
                    "@mfr": "MfrConnectionName"
                    "@temp": "TConnectionName"
            },
            "OutputPortName": {
                    "@temp": "TComponentNameOutputPortName"
            }
}
```

(iv) Export dck

When all files needed are loaded and/or exported the dck-file (TRNSYS) can be built. Either this is done without launching a simulation directly afterwards to just generate the file by pressing the respective button in the tool bar

or a simulation is launched directly through hitting

CHAPTER 4

Running simulations

pytrnsys runs TRNSYS simulations based on configuration files with the extension `.config`. The general idea behind this is to provide a fast and easily accessible way to define, run and analyse both single simulations as well as parametric studies.

Note: The configuration file does not require a header. It should contain different keyword commands on single lines. Comments start with # characters. End of line comments are possbile.

The existing commands are listed per topic in the following.

4.1 Simulation control

`outputLevel` Output message level according to the logging package. (Options: "DEBUG", "INFO" (default), "WARNING", "ERROR", and "CRITICAL"):

```
string outputLevel "INFO"
```

`ignoreOnlinePlotter` If set to True, the TRNSYS online plotters are commented out in all the dck-files. No online plotters are shown during the simulation run. The TRNSYS progress bar window is still displayed (default: False):

```
bool ignoreOnlinePlotter False
```

`removePopUpWindow` Online plotters as well as the progress bar window are suppressed during the simulations, which corresponds to the TRNSYS hidden mode (default: False):

```
bool removePopUpWindow False
```

`reduceCpu` When running simulations in parallel on different logical cores of your CPU, you can specify how many cores will be left unoccupied (default: 0):

```
int reduceCpu 0
```

checkDeck If set to True, during merging the ddck-files, the specified and given amount of Equations and Parameters in each block are checked for inconsistencies (default: True):

```
bool checkDeck True
```

parseFileCreated Saves the parsed dck-file that can be used to locate the line where `checkDeck` found errors (default: True):

```
bool parseFileCreated True
```

runCases If set to False, the dck-files are created and saved in the normal structure but not executed (default: True):

```
bool runCases True
```

doAutoUnitNumbering If set to True, the units of the merged dck-file are renumbered to avoid duplicates. This parameter should usually be set to the default True:

```
bool doAutoUnitNumbering True
```

generateUnitTypesUsed If set to True, a file called `UnitType.info` containing the TRNSYS-Type numbers used is saved in the main run-folder (default True):

```
bool generateUnitTypesUsed True
```

addAutomaticEnergyBalance If set to True, an automatic energy balance printer is created in the dck file. For more information see *Default plotting for TRNSYS results* (default: True):

```
bool addAutomaticEnergyBalance True
```

nameRef Base name of the dck-file(s) created. Default base name is "pytrnsysRun":

```
string nameRef "pytrnsysRun"
```

rerunCases TBD:

```
bool rerunCases False
```

runFromCases TBD:

```
bool runFromCases False
```

copyBuildingData TBD:

```
bool copyBuildingData False
```

runType TBD (Options: "runFromConfig" (default), "runFromCases", "runFromFolder"):

```
string runType "runFromConfig"
```

pathToConnectionInfo Specify the path to the DdckPlaceHolderValues.json, which you would like to use to replace the placeholders in the ddck-files. It overrules the behaviour of replacing the placeholders with the defaults:

```
string pathToConnectionInfo "path-to-your-project-folder\DdckPlaceHolderValues.
↪json"
```

4.2 Tracking

trackingFile When running multiple simulations the status of each simulation can be tracked with the help of a json-file. When a simulation is started, this is entered with a timestamp into this file. Once the simulation is finished this entry will be overwritten accordingly. Like this one can keep track of which simulations were aborted (for whatever reason) after having been launched. To activate this functionality you need to specify the full path of the json-file to be created:

```
string trackingFile "...\[name].json"
```

masterFile If several simulations are run from different instances, the tracking can be taken one step further by employing a "master-file" in the form of a csv-file. It also tracks the status of different simulations based on the tracking json-files. One important feature is that, when it is used, simulations (identified by the name of the dck-file) that are already entered as a "success" won't be run again. This is useful for redoing parametric studies where single simulations failed. If this is the case one can do the needed corrections and then simply launch the same parametric study again and the "master-file" will ensure that no unnecessary repetitions of simualtions are executed. To activate this functionality you need to specify the full path of the csv-file to be created:

```
string masterFile "...\[name].csv"
```

4.3 Paths

trnsysExePath Specify the path to the exe-file of TRNSYS, which you would like to use to run the simulations:

```
string trnsysExePath "C:\TRNSYS18\Exe\TrnEXE.exe"
```

pathBaseSimulations If specified, the location of where the simulation is run is changed to the given path. It overrules the normal behavior of executing the simulations in the current working directory:

```
string pathBaseSimulations "path-to-your-simulation-folder"
```

addResultsFolder Specify the path to which you would like to save your simulation results:

```
string addResultsFolder "path-to-your-results-folder"
```

Definition of path alias You can define an alias for a path to be used in a different place. If, e.g., you want to load many ddck-files from "C:\GIT\pytrnsys\data\ddcks" you can give this path an alias such as "PYTRNSYS$". The "$" at the end of the alias needs to be included always to mark it as such:

```
string PYTRNSYS$ "C:\GIT\pytrnsys\data\ddcks"
```

4.4 Scaling

scaling If this is set to "toDemand" the scaling functionality is activated for the parameter variation. (Options: "False" (default), "toDemand"):

```
string scaling "False"
```

scaleHP Specify the size of the heat pump of the system in kW through some numerical value `x` and some `variable` that is defined in the `scalingReference` file (see below):

```
string scaleHP "x*variable"
```

scalingVariable This defines a variable from the `scalingReference` file (see below) that is used for scaling the parameter variations:

```
string scalingVariable "name-of-your-scaling variable"
```

scalingReference Full path to a json-file containing the variables used for scaling (see above):

```
string scalingReference "...\[name].json"
```

4.5 Parameter variation

A core feature of pytrnsys is the parameter variation. pytrnsys allows either to modify TRNSYS simulation parameters in the configuration file statically or with variations that result in parametric runs.

deck A certain `trnsysVariable` defined in the dck-file can be set to a certain `value` by overwriting the previous one through:

```
deck trnsysVariable value
```

This feature can for example be used to change the starting time of the simulation(s):

```
deck START 4344
```

variation A parametric study, i.e. several TRNSYS simulations with different `values` for a certain `trnsysVariable`, can be launched with the following command:

```
variation variationName trnsysVariable value1 value2 value3 ...
```

Here, `variationName` defines how the variation will be noted in the names of the dck-files to be generated. In general the `values` are absolute values of the respective `trnsysVariable`. If *scaling* is set to "toDemand", however, then the `values` are the factors by which the `scalingVariable` is mulitplied to receive the actual numerical value of the `trnsysVariable`. If, e.g., the `scalingVariable` is the yearly heat demand of a system in MWh and the `trnsysVariable` to be varied is the area of the solar collector field called `AcollAp` in m2, then this area can be varied as multiples of the yearly heat demand (in m2/MWh) like this:

```
variation Ac AcollAp 1.0 1.5 2.0 2.5 3.0
```

combineAllCases If several `variations` are defined, this parameter controls their combination. If it is set to the default True, all combinations are created. So if n values are given for variation 1 and m values are in variation 2 the total amount of simulations executed will be (m x n). If it is set to False, the amount of values of all variations has to be equal and they are combined according to their order:

```
bool combineAllCases True
```

changeDdckFile Instead of only varying one or more variable, due to its modular nature, pytrnsys also allows to vary through different ddck-files:

```
changeDDckFile originalDdck ddckVariation1 ddckVariation2 ddckVariation3 ...
```

This can be used, e.g., to change the weather location for a simulation very swiftly. Assuming the weather data is specified through a file of the type `City..._dryK` and the locations `BAS`, `CDF` and `LUG` should be simulated, this can be done with the command:

```
changeDDckFile CityBAS_dryK CityBAS_dryK CityCDF_dryK CityLUG_dryK
```

4.5.1 random variations

If the influence of many different parameters is of interest, random variations might be needed. Random variations can be done with the keywords `randvar`, `randvarddck`, `nrandvar` and `randseed`. If using random variations, do not include any regular variations and be aware, that only one ddck can be changed with `randvarddck`.

`randvar` Random variations of trnsys constants can be executed by adding one or multiple lines like the following:

```
randvar variationName trnsysVariable minValue maxValue stepSize
```

Here, `variationName` defines how the variation will be noted in the names of the dck-files to be generated. The `minValue` and `maxValue` are the minimum and maximum Value that the `trnsysVariable` will take, respectively. `stepSize` describes the step size of the values that can be taken between `minValue` and `maxValue`. Make sure that `maxValue` = `minValue` + `n` * `stepSize`. Where `n` is an integer. In the following example, the storage size will be taken between 0.5 m3 and 1 m3 in steps of 0.1, so it will have the options 0.5, 0.6, 0.7, 0.8, 0.9, 1:

```
randvar Vtes storageSize 0.5 1 0.1
```

`randvarddck` Random variations of ddcks can be included with the following command:

```
randvarddck originalDdck ddckVariation1 ddckVariation2 ddckVariation3 ...␣
↪ddckVariationn
```

For every iteration pytrnsys then takes randomly one of the n `ddckVariation` instead of the `originalDdck`, that is specified in the used ddcks section. Currently, only one ddck can be randomly varied.

`nrandvar` This keyword describes the total number of random variations to be simulated, if e.g. 1000 variations should be simulated, the following line has to be added:

```
nrandvar 1000
```

Default value of nrandvar is 100.

`randseed` This keyword describes an integer, that is used as a seed. If a seed is set, then rerunning a simulation will yield the same variations. A different integer will yield different random variations.

Example:

```
randseed 1
```

If randseed is not defined or of it is set to None, then the variations will be different every time.

4.6 ddck files

The core of the run configuration file is the ddck section. In this part of the configuration file, the different modular ddck files that should be used in the simulation are specified:

```
PATH_ALIAS_1$ head
PATH_ALIAS_2$ ddck_1
PATH_ALIAS_2$ ddck_2
...
```

(continues on next page)

(continued from previous page)

```
PATH_ALIAS_m$ ddck_n
PATH_ALIAS_1$ end
```

An example can be found in the example section below. The path to the repository root can be either absolute or relative. If a relative path is detected, pytrnsys will interpret it as relative to the configuration file location.

4.7 Example

Here is an example of a run configuration file. It is taken from the example project solar_dhw (`run_solar_dhw.config`):

```
######### Generic ########################
bool ignoreOnlinePlotter  True
int reduceCpu  4
bool parseFileCreated True
bool runCases True
bool checkDeck True

############ AUTOMATIC WORK BOOL#################################

bool doAutoUnitNumbering True
bool generateUnitTypesUsed True
bool addAutomaticEnergyBalance True

############PATHS#################################

string trnsysExePath "C:\Trnsys17\Exe\TRNExe.exe"
string addResultsFolder "solar_dhw"
string PYTRNSYS$ "..\..\pytrnsys_ddck\"
string LOCAL$ ".\"

################SCALING#########################

string scaling "False" #"toDemand"
string nameRef "SFH_DHW"
string runType "runFromConfig"

############PARAMETRIC VARIATIONS#################

bool combineAllCases True
variation Ac AcollAp 2 3 4 6 8 10
variation VTes volPerM2Col 75 100

############FIXED CHANGED IN DDCK#################

deck START 0    # 0 is midnight new year
deck STOP  8760 #
deck sizeAux 3

############USED DDCKs#################

PYTRNSYS$ generic\head
PYTRNSYS$ demands\dhw\dhw_sfh_task44
PYTRNSYS$ weather\weather_data_base
```

(continues on next page)

(continued from previous page)

```
PYTRNSYS$ weather\SIA\normal\CitySMA_dryN
PYTRNSYS$ solar_collector\type1\database\type1_constants_CobraAK2_8V
PYTRNSYS$ solar_collector\type1\type1
LOCAL$ solar_dhw_control
LOCAL$ solar_dhw_storage1
LOCAL$ solar_dhw_hydraulic
LOCAL$ solar_dhw_control_plotter
PYTRNSYS$ generic\end
```

CHAPTER 5

Post-processing simulations

pytrnsys post-processes TRNSYS simulations based on configuration files with the extension `.config`, just like when *running them*.

Note: The configuration file does not require a header. It should contain different keyword commands on single lines. Comments start with # characters. End of line comments are possbile.

The existing commands are listed per topic in the following.

5.1 Generic

`typeOfProcess` (string, default 'completeFolder')

This parameter defines how data is processed. There are various possible arguments:

- 'completeFolder': Identifies data sets through lst files in `pathBase` and its subfolders.
- 'individual': Only the specified files will be processed. With `timeBase` specifying the time steps in the file (`monthly`, `daily`, `hourly`, or `timeStep` for a user-defined time base) and `dataFolder` the path to the file, which needs to be defined somewhere else in the config file, the files are specified like:

```
stringArray fileToLoad "timeBase" "dataFolder" "fileName"
```

- 'casesDefined': TBD
- 'citiesFolder': Data sets are defined by subfolders in `pathBase`, which are named according to the `cities` parameter.
- 'config': TBD
- 'json': Identifies data sets through json files in `pathBase` and its subfolders. Can also be used on `-results.json` files.

`processParallel` (bool, default True) If set to True, pytrnsys will process the simulation sub-folders in parallel. The amount of parallel processes will be the total amount of CPUs minus `reduceCpu`.

`processQvsT` (bool, default True) Flag to disable the QvsT processing. Since this is computationally very expensive it can be useful to disable the QvsT plots if not needed.

`cleanModeLatex` (bool, default False) If set to True, all plot files will be deleted after they are collected in the results pdf-file. If set to False, they will remain in the simulation subfolder.

`forceProcess` (bool, default True) If set to False, allready processed folders will not be processed again.

`plotStyle` (string, default 'line') If set to 'dot', dots will be used instead of lines for the respective plots.

`setPrintDataForGle` (bool, default True) Print the Data of the plots for further use in GLE plots.

`figureFormat` (string, default 'pdf') Format in which the plots of the processing will be saved. All formats that are supported by *matplotlib.pyplot.savefig <https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.savefig.html>* are supported

`plotEmf` (bool, default False) If set to true, all plots will be exported in the emf format. Requires Inkscape.

5.2 Paths

`latexNames` (string) Path to the latexNames json-file. Can either be an absolute path or a path relative to the configuration file. If not specified, the default latexName json-File of pytrnsys is used.

`pathBase` (string) Path of the folder to be processed. If not specified, the current working directory is used instead.

`inkscape` (string) Path of the Inkspace executable. Required for using *plotEmf <ref-plotEmf>*.

5.3 Time selection

Pytrnsys is designed to process one full year. If more than a year is simulated, the months that are used for processing have to be specified.

`yearReadedInMonthlyFile` (int, default -1) Year of the simulation that is used for processing. 0 is the first year, 1 the second year and so on. If the value is set to -1 pytrnsys will use the last 12 months of the simulation for processing.

`firstMonth` (["January", "February", "Mach", . . . , "December"], default "January") Month in the chosen year where the 12-month processing period begins. If the value is e.g. "November" November to October will be analysed.

5.4 Processing TRNSYS data

During processing pytrnsys reads in the following values automatically:

1. All parameter and equation variables that are statically defined in the dck.file. Pytrnsys recursively detects static variables by checking for any type outputs in the variables involved.
2. All monthly printer values of the simulation. The pytrnsys ddcks save all printer files in the temp folder inside the directory where the simulation is executed. If custom printers are defined, the same location is required.
3. All hourly printer values of the simulation.

All values can be adressed in the config file by their name in the header of the trnsys printer file. It is recommended to duplicate the internal TRNSYS name in the header of the printer.

Note: While TRNSYS is not case sensitive, Python is. So be careful about upper and lower cases during post processing. If the string in the configuration file does not match the header of the printer file or the TRNSYS name of the static parameter in the dck-file, pytrnsys will not be able to find the value and throw a key-error.

When TRNSYS data is read in, pytrnsys will automatically create some variables. These are:

4. From monthly values of `foo` the total sum over the simulated period is calculated and can be called by `foo_Tot`. Furthermore, `Cum_foo` is created, which is an array of the accumulated values of `foo` over the months.
5. From hourly values of `bar` the minimum, maximum and average values over the simulated period are calculated, which can be called by `bar_Min`, `bar_Max` and `bar_Avg`.

5.5 Processing generic data

To process generic data, add the following expression to the header of your configuration file:

```
bool isTrnsys False
```

You then need to specify how pytrnsys should access your data. One way is to identify a data set with a json file that includes the parameters of the data set in the format of a python dictionary. When you have such a json in each data set folder, you should use:

```
string typeOfProcess "json"
```

Furthermore, you need to specify the folder (here, e.g.: `dataFolder`) containing your data sets with:

```
string pathBase "..\dataFolder"
```

The program will look for json-files in `dataFolder` and on each subfolder level. It will then load csv-files, which are in the same folders as the json-files it found. At the moment it can load hourly, daily, and monthly data. The names of the respective csv-files need to contain the keywords `_Stunden`, `_Tage`, or `_Monat`.

5.6 Calculations

In the processing-configuration file, the user can specify custom calculations based on the TRNSYS results that were read in and the values that are calculated by default. The type of each equation has to be defined by a key word that tells pytrnsys what values should be used. This is necessary since some variables could be both in an hourly as well as a monthly printer. The following calculation keywords are available:

`calc` Calculates a new scalar value out of other scalar values such as static TRNSYS parameters or yearly sums or hourly maxima.

`calcMonthly` Calculates new monthly values (array with length 12) out of other monthly values or scalar values.

`calcDaily` Calculates new daily values (array with length 365) out of other hourly values or scalar values.

`calcHourly` Calculates new hourly values (array with length 8760) out of other hourly values or scalar values.

`calcMonthlyFromHourly` Calculates new monthly values (array with length 12) out of hourly values or scalar values.

A calculations section could be of the following structure. A full working example can be found in the example below:

```
calc alpha = foo_Tot/bar_Max
calcMonthly = foo/foo_Tot*1000
calcHourly = (bar+100)**2
```

acrossSetsCalc Can execute calculations across data sets with variables from the results json-files. Equations are provided as arguments and indicated by a = and conditions by `:` and stated as `key:value`. A function call (optional arguments in square brackets) then looks like:

```
stringArray acrossSetsCalc "x_variable" "y_variable" "calculation variable"
↪"equation 1" ["equation 2"] ... ["key 1:value 1"] ["key 2:value 2"] ...
```

Here `calculation variable` is a key of the results json-files and specifies what arguments can go into an equation. An example for an equation looks like:

```
nameOfValueToBeCalculated=(foo+bar)*100
```

where `foo` and `bar` are valid values of the `calculation variable`. The program will take different data sets with the same `x`- and `y`- but different `calculation variable`-values and execute the equation for these. Hence, you need to ensure that these combination exist in your data sets. A csv with the calculated results will be generated.

5.7 Results file

For further custom processing of the simulation results, required scalar and monthly values can be saved to a results json-file.

results Determines which variables should be stored in a dedicated json-file for each data set:

```
stringArray results "variable 1" "variable 2" ...
```

jsonInsert Adds `value` as `parameter name` to the generated `-results.json` files:

```
stringArray jsonInsert "parameter name" "value"
```

pathInfoToJson Scans the paths of the generated `-results.json` files for keywords and adds them as the respective `parameter name` in said json-files, and adds an empty string, if it doesn't find any of the keys in the respective path:

```
stringArray pathInfoToJson "parameter name" "key 1" "key 2" ...
```

jsonCalc Allows to do calculations with the variables saved in `-results.json`, of which the results are then saved to the respective json-file as whatever is given as the variable name on the left side of the =:

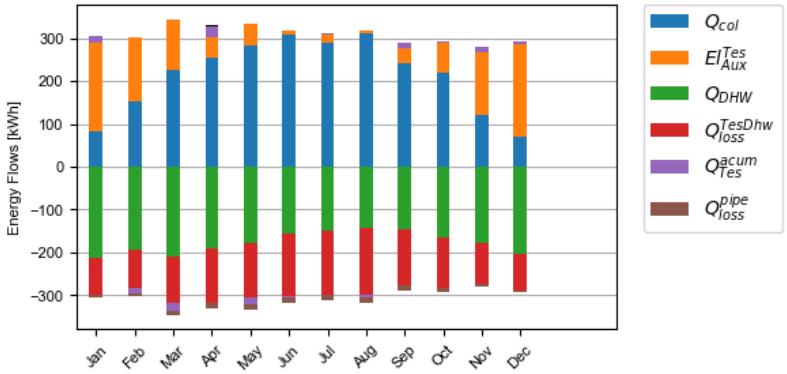
```
stringArray jsonCalc "newVariable1=rightSideOfEquation1"
↪"newVariable2=rightSideOfEquation2" ...
```

5.8 Plotting

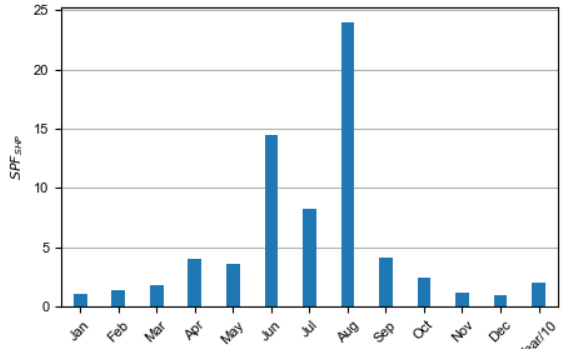
5.8.1 Default plotting for TRNSYS results

By default the processing creates a pdf with the following content:

1. A table displaying the total simulation time and the number of iteration errors.
2. A table with the monthly heat balance. The values are also shown in a plot, in the case of the solar domestic hot water example system this looks like the following:



3. A electricity balance similar to the heat balance.
4. The system seasonal performance factor both in a table and a plot. Again, the SPF plot of the solar domestic hot water system looks like:

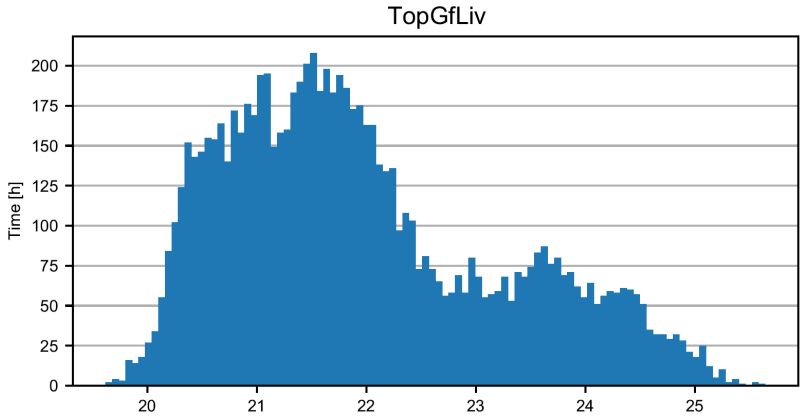


5.8.2 Plotting hourly data

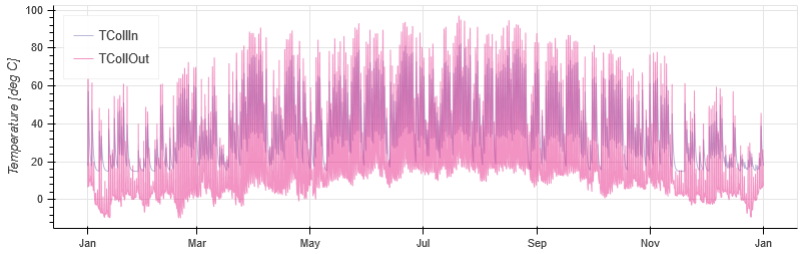
Note: If an argument in the code excerpts below is set in square brackets, it is optional.

`plotT` This generates one or more frequency analysis plots for hourly variables, i.e., bar plots of bandwidth bins for certain values of the respective variables (originally aimed at temperatures, hence the name). It provides an overview over how often a certain value range of a variable appears:

```
stringArray plotT "hourly variable 1" "hourly variable 2" ...
```

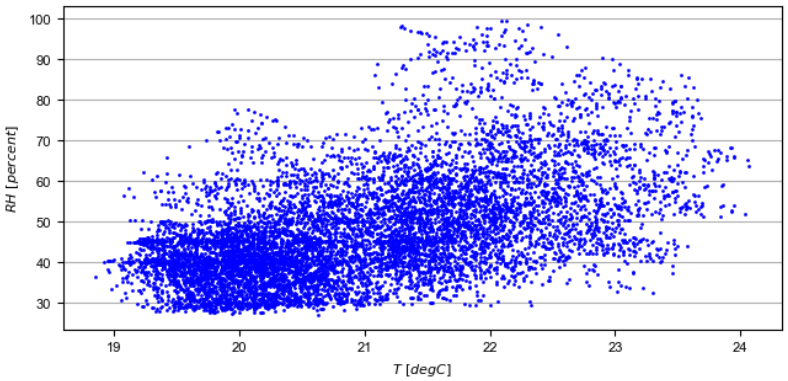


plotHourly Hourly printed values can be displayed in an interactive HTML plot that is created using the bokeh plotting library.



scatterHourly Hourly printed values can be displayed as a scatter plot:

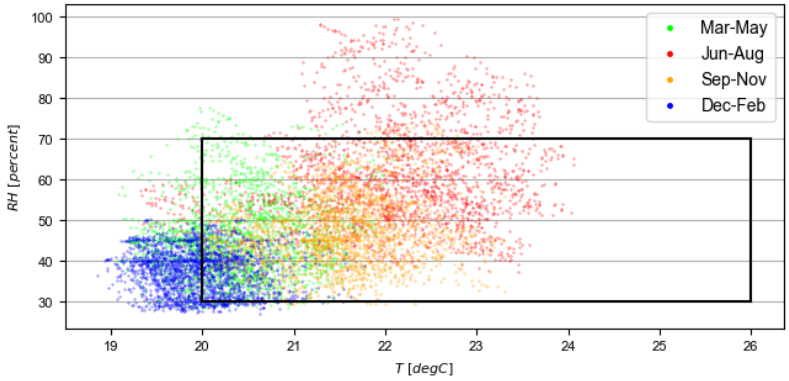
```
stringArray scatterHourly "x_variable" "y_variable"
```



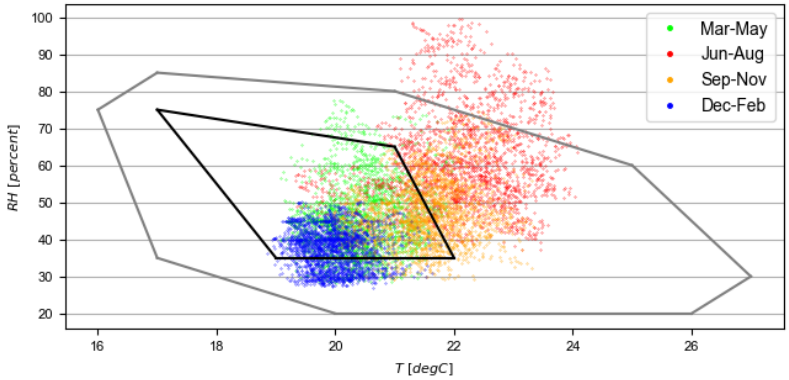
comfortHourly The hourly printed humidity of a room can be plotted against the hourly printed room temperature and be compared to different comfort norms:

```
stringArray comfortHourly ["norm"] "temperature_variable" "humidity_variable"
```

There are two norm boundaries available. The default one (can also be actively called by setting `norm` to `ISO7730`) is ISO 7730:



The alternative one is according to this publication and can be employed by setting `norm` to `Dahlheimer`:



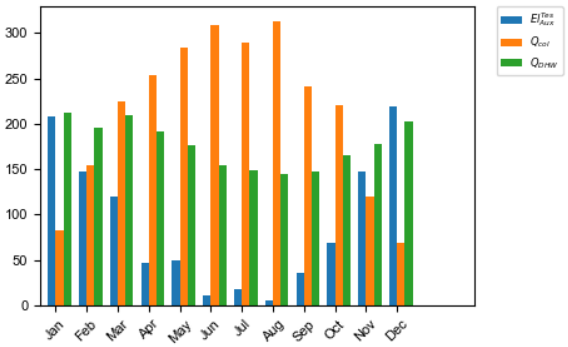
`plotHourlyQvsT` Adds a cumulative plot that contains a line for each heat temperature pair given in the string array. Used to show at what temperature levels the heat is released or consumed in different system components. Uses hourly printer files.

5.8.3 Plotting monthly data

Note: If an argument in the code excerpts below is set in square brackets, it is optional.

`monthlyBars` Plots a monthly bar plot that shows all variables grouped side by side. The name of the pdf to be created needs to specified through `pdf name`:

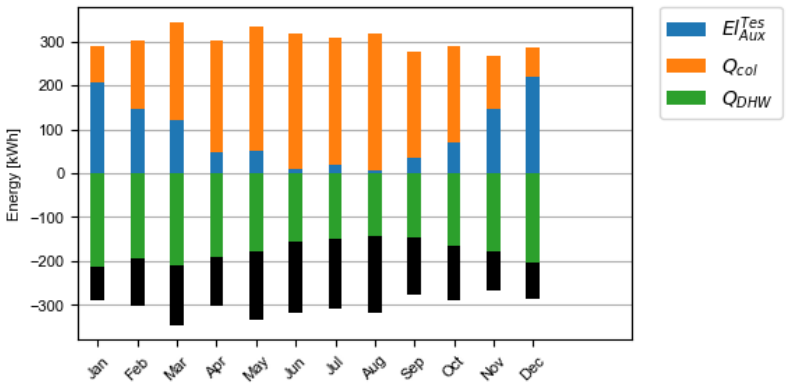
```
stringArray monthlyBars "pdf name" "variable 1" "variable 2" ...
```



monthlyBalance Custom monthly balance. The sign of the values can be inverted by adding a - in front of the variable name. If positive and negative values don't add up to zero, the imbalance is shown as black bars. The name of the pdf to be created needs to be specified through `pdf name`. When adding the optional `style:relative` the bars will be shown as values relative to the positive sum of the monthly energy values:

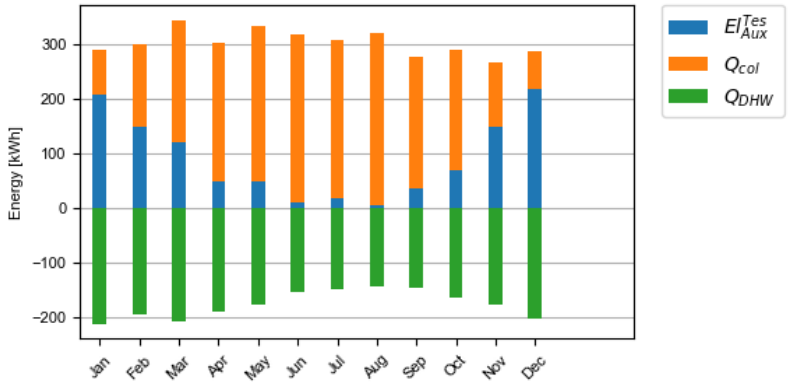
```
stringArray monthlyBalance "pdf name" ["style:relative"] "variable 1" "variable 2
↪" ...
```

In the solar domestic hot water example system this can be demonstrated by plotting the two system inputs Q_{col} and El_{Aux}^{Tes} and the usable output of the domestic hot water demand. The imbalance in this case are the overall losses of the system.



monthlyStackedBar Similar to the `monthlyBalance` but without showing the imbalance. The name of the pdf to be created needs to be specified through `pdf name`:

```
stringArray monthlyStackedBar "pdf name" "variable 1" "variable 2" ...
```



5.8.4 Plotting time-step data

`plotTimestepQvsT` Adds a cumulative plot that contains a line for each heat temperature pair given in the string array. Used to show at what temperature levels the heat is released or consumed in different system componenets. Uses timestep printer files.

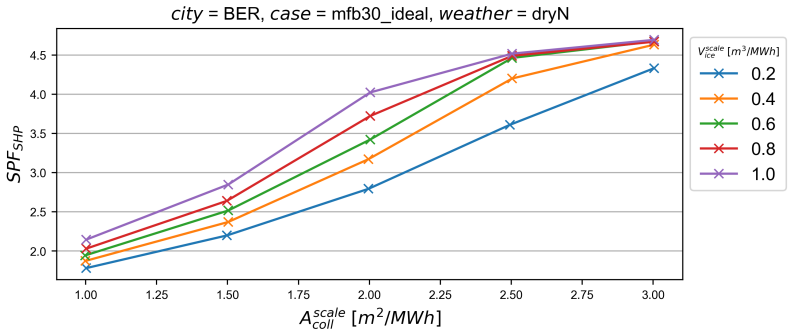
5.8.5 Plotting parametric data

Note: If an argument in the code excerpts below is set in square brackets, it is optional.

Note: All variables used for the parametric plots need to be saved in the `-results.json` files.

`comparePlot` When processing parametric runs, scalar results of the simulations can be visualized in comparison plots. The first variable of the string array is shown on the x-axis. The second variable is shown on the y-axis. The third is represented as different lines, and the fourth as different marker styles:

```
stringArray comparePlot "x_variable" "y_variable" ["series 1 variable"] ["series␣
↪2 variable"] ["filter1"] ["filter2"] ...
```



Additionally, you can filter the data that should be plotted by passing in filter expressions for the "filter"s above: only the data taken from `-results.json` files that match the filter expressions will then be considered. Filter expressions can take the following form:

Equality:

```
key=value
key=value1|value2|...
```

For multiple values to be included, they need to be separated by | without spaces. For equalities the values can be numbers or strings, depending on the type of the key.

Inequality:

```
key>value
key<value
key>=value
key<=value
```

Logically, for inequalities value needs to be a number.

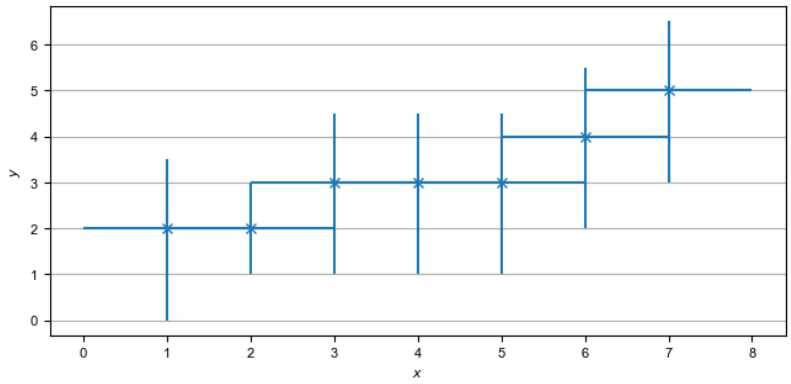
Ranges:

```
value1<key<value2
value1<key<=value2
value1<=key<value2
value1<=key<=value2
```

Ranges need to be specified by < or <= and the values need to be numbers. Note that each key can only be used once, so a range cannot be replaced by two separate inequality statements.

comparePlotConditional (***deprecated***) Same as comparePlot, only retained for backwards compatibility. Use comparePlot instead.

comparePlotUncertain Same as comparePlot but displays uncertain values with error bars:

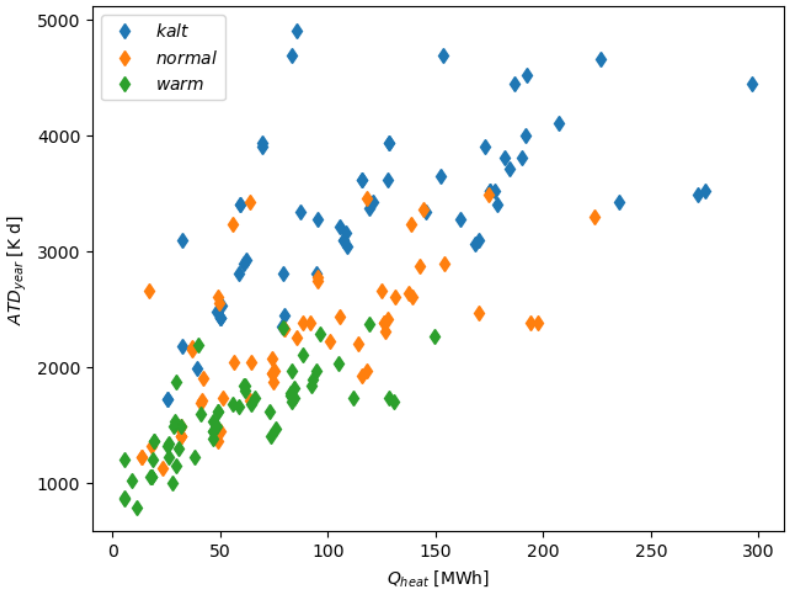


acrossSetsCalculationsPlot Has the same basic functionality as acrossSetsCalc, but can plot the results of equations provided:

```
stringArray plotCalculationsAcrossSets "x_variable" "y_variable" "calculation␣
↪variable" "equation 1" ["equation 2"] ... ["key 1:value 1"] ["key 2:value 2"] ..
↪.
```

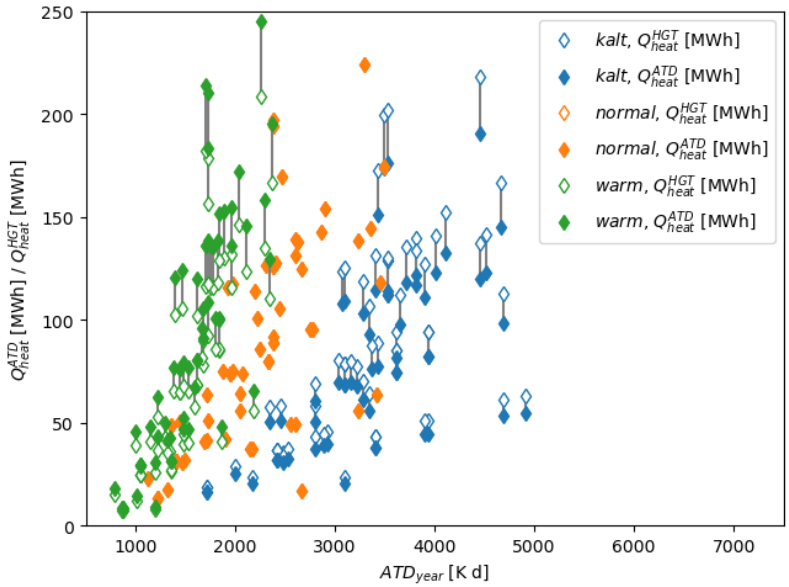
scatterPlot Generates scatter plots:

```
stringArray scatterPlot "x_variable" "y_variable" ["series 1 variable"]
```



When a – is added to `y_variable` a scatter plot indicating differences is generated:

```
stringArray scatterPlot "x_variable" "y_variable 1-y_variable 2" ["series 1␣
↪variable"]
```



5.9 Example

The following processing-configuration file is part of the solar domestic hot water example system:

```
######### Generic ########################
bool processParallel False
bool processQvsT True
bool cleanModeLatex False
```

(continues on next page)

(continued from previous page)

```
bool forceProcess  True
bool setPrintDataForGle True
bool printData True
bool saveImages True
int reduceCpu 1

######### Time selection ########################
int yearReadedInMonthlyFile -1
int firstMonthUsed 6     # 0=January 1=February 6=July 7=August

############# PATHS ##############################
string latexNames ".\latexNames.json"
string pathBase "C:\Daten\OngoingProject\pytrnsysTest\SolarDHW_newProfile"

############# CALCULATIONS ##############################

calcMonthly fSolarMonthly = Pcoll_kW/Pdhw_kW
calc fSolar = Pcoll_kW_Tot/Pdhw_kW_Tot

calcMonthly solarEffMonthly = PColl_kWm2/IT_Coll_kWm2
calc solarEff = PColl_kWm2_Tot/IT_Coll_kWm2_Tot

############# CUSTOM PLOTS ##############################
stringArray monthlyBars "elSysIn_Q_ElRot"  "qSysIn_Collector" "qSysOut_DhwDemand"
stringArray monthlyBars "solarEffMonthly"
stringArray monthlyBalance "elSysIn_Q_ElRot"  "qSysIn_Collector" "-qSysOut_DhwDemand"
stringArray monthlyStackedBar "elSysIn_Q_ElRot" "qSysIn_Collector" "-qSysOut_DhwDemand
↪"

stringArray plotHourly "Pcoll_kW" "Pdhw_kW" "TCollIn" "TCollOut"  # "effColl" #␣
↪values to be plotted (hourly)
stringArray plotHourlyQvsT "Pdhw_kW"  "Tdhw" "Pcoll_kW" "TCollOut"

stringArray comparePlot "AcollAp" "fSolar" "volPerM2Col"
stringArray comparePlot "AcollAp" "fSolar" "volPerM2Col"
stringArray comparePlot "AcollAp" "Pdhw_kW_Tot" "volPerM2Col"


############# RESULTS FILES ##############################
stringArray hourlyToCsv "CollectorPower" "IT_Coll_kWm2" "PColl_kWm2"
stringArray results  "AcollAp"  "Vol_Tes1"   "fSolar"  "volPerM2Col"  "Pdhw_kW_Tot" #␣
↪values to be printed to json
```

CHAPTER 6

Structure of a ddck-file

The file sctructure of the ddck files is shown in the example.ddck file in the pytrnsys_ddck repository root. Each ddck should contain the following parts. Some parts can be empty if not used.

6.1 Header

In the header the name of the ddck as well as a contact persion, the creation date and some information about the last changes are specified. In addition, there is a section for general descriptions about the ddck:

```
*******************************
**BEGIN example.ddck
*******************************

****************************************
** Contact person : author
** Creation date  : date
** Last changes   : date, name
****************************************

************************************************************************
** Description:
** Overall description of the ddck file
** TODO: Any improvements needed
************************************************************************
```

6.2 Inputs from hydraulic solver

The pytrnsys ddcks are designed to be used in combination with the hydraulic solver type 935. In each timestep the hydraulic solver computes all mass flows and component input temperatures starting depending on the component output temperatures, the pump powers and the valve positions. Therefore, the mass flows and component input temperatures are outputs of the hydraulic solver that have to be connected to the component ddcks as inputs:

```
**********************************
** inputs from hydraulic solver
**********************************

** tIn from hydraulic
** tIn from hydraulic with placeholder:
** tIn = @temp(PortName, DefaultPipeName)
** mIn from hydraulic
** mIn from hydraulic with placeholder:
** mIn = @mfr(PortName, DefaultPipeName)
```

6.3 Outputs to the hydraulic solver

Similar to the inputs, the output temperatures of the component should be stated here, such that they are easily accessible to be connected to the hydraulics file:

```
**********************************
** outputs to hydraulic solver
**********************************

** which outputs will be used to connect the hydraulic solver
** typically tOutType will be defined here to be used in the hydraulic ddck
** tOutType with placeholder:
** @temp(PortName, DefaultName) = tOut
```

6.4 Outputs to the energy balance

In the processing, pytrnsys automatically computes the systems heat and electricity energy balance. All variables that should be collected for the energy balance have to be specified in this section according to the right nomenclature:

```
*******************************************************************************************
** outputs to energy balance in kWh and ABSOLUTE value
** Following this naming standard : qSysIn_name, qSysOut_name, elSysIn_name, elSysOut_
↪name
*******************************************************************************************

** Add here those variables that will go into the overall energy balance of the system
** These values will be used to automatically generate the energy balance
```

6.5 Dependencies with other ddck files

In order to enhance modularization, dependencies with other ddcks should be kept minimal. Dependencies that cannot be avoided and are neither part of the component-database relation or the general variables should be declarated and reassigned to an internally used variable in this part:

```
**********************************
** Dependencies with other ddck
**********************************

```

(continues on next page)

(continued from previous page)

```
** Re-assing here the variables necessary from other types
** variableInternal = variableExternal
** Exception: those from general variables
```

6.6 Outputs to other ddck files

Variables that are designated to be used in other ddck files should be added here:

```
**********************************
** outputs to other ddck
**********************************

** Add here the outputs of the TYPE or TYPES that will be used in other types
** Exception: those for printers and so on dont need to be here.
```

6.7 Precalculations related to parameter scaling and pre-processing

Usually, in the declaration of a TRNSYS component, many parameters are calculated out of more general system variables. All calculations to determine the right parameters inputs for the type go here:

```
**********************************
** Begin CONSTANTS
**********************************
```

6.8 Type section

TRNSYS has its own syntax that calls the type dll files. This core part of the ddck goes here:

```
**********************************
** Begin TYPE
**********************************
```

6.9 Component printers

Each component should have a monthly as well as an hourly printer. This helps to simplify the setup and the processing of the simulation. In addition, an online plotter is a nice tool for the debugging of the system:

```
**********************************
** Monthly printer
**********************************

**********************************
** Hourly printer
**********************************

**********************************
```

(continues on next page)

(continued from previous page)

```
** Online plotter
***********************************
```

6.10 Hydraulics files

The hydraulics file represents the systems hydraulics layout. Each pytrnsys example system except the pv battery system has its own hydraulic layout file. In order to create your own hydraulic files that represent the hydraulics of your choice you need access to the pytrnsys GUI. The hydraulics file are not part of the ddck repository. The hydraulic files of the example systems are located in the example system folder of **pytrnsys_examples**.

6.11 Examples

The following example shows the ddck file of the solar collector type 1 used in the solar domestic hot water system:

```
*******************************
**BEGIN Type1.ddck
*******************************

****************************************
** Contact person : Dani Carbonell
** Creation date  : 10.01.2010
** Last changes   : 03.2020 Jeremias Schmidli
****************************************

************************************************************************
** Description:
** Collector model using efficiency curve efficiency
************************************************************************

**********************************
** inputs from hydraulic solver
**********************************

EQUATIONS 2
TCollIn = TPiColIn
MfrColl = ABS(MfrPiColIn)

**********************************
** outputs to hydraulic solver
**********************************

EQUATIONS 1
TCollOut = [28,1]

**********************************
** outputs to other ddck
**********************************

*********************************************************************************************
** outputs to energy balance in kWh and ABSOLUTE value
** Following this naming standard : qSysIn_name, qSysOut_name, elSysIn_name, elSysOut_
↪name
```

(continues on next page)

(continued from previous page)

```
**********************************************************************************************

EQUATIONS 1
qSysIn_Collector = PColl_kW

**********************************
** Dependencies with other ddck
**********************************

EQUATIONS 1
pumpColOn = puColOn

CONSTANTS 2
C_tilt = slopeSurfUser_1  ! @dependencyDdck Collector tilt angle / slope [°]
C_azim = aziSurfUSer_1    ! @dependencyDdck Collector azimuth  (0:s, 90:w, 270: e) [°]

EQUATIONS 4
**surface-8
IT_Coll_kJhm2 = IT_surfUser_1  ! Incident total radiation on collector plane, kJ/hm2
IB_Coll_kJhm2 = IB_surfUser_1  ! incident beam radiation on collector plane, kJ/hm2
ID_Coll_kJhm2 = ID_surfUser_1  ! diffuse and ground reflected irradiance on collector␣
↪tilt
AI_Coll = AI_surfUser_1  ! incident angle on collector plane, °

EQUATIONS 5
IT_Coll_kW = IT_Coll_kJhm2/3600     ! Incident total radiation on collector plane, kW/
↪m2
IB_Coll_kW = IB_Coll_kJhm2/3600     ! incident beam radiation on collector plane, kW/
↪m2
ID_Coll_kW = ID_Coll_kJhm2/3600     ! diffuse and ground reflected irradiance on␣
↪collector tilt (kW/m2)
IT_Coll_Wm2 = IT_surfUser_1/3.6
IT_Coll_kWm2 = IT_surfUser_1/3600

**********************************
** Begin CONSTANTS
**********************************

CONSTANTS 3
MfrCPriSpec = 15  ! Coll. Prim. loop spec. mass flow [kg/hm2]
AcollAp=5         ! Collector area
MfrCPriNom = MfrCPriSpec*AcollAp !

**********************************
** Begin TYPE
**********************************

UNIT 28 TYPE 1
PARAMETERS 11
nSeries       ! number in series
AcollAp       ! collector area
cpBri          ! fluid specific heat kj(kgK
efficiencyMode ! efficiency mode
testedMfr      ! tested flow rate kg/(hm2)
Eta0          ! intercept efficiency
a1            ! efficiency slope kJ/hm^2K
a2            ! efficiency curvature kJ/hm^2K^2
```

(continues on next page)

(continued from previous page)

```
2             ! optical mode
FirstOrderIAM  ! 1st order IAM
SecondOrderIAM ! 2nd order IAM
INPUTS 9
TCollIn
MfrColl
Tamb
IT_Coll_kJhm2
IT_H
ID_Coll_kJhm2
0,0
AI_Coll !Flo check ! JS: This was defined wrong before (C_azim, even though it is␣
↪incident angle input). Now it should be correct.
C_tilt !Flo check  ! JS: This should be correct
*** INITIAL INPUT VALUES
20 0 10 0 0 0 GroundReflectance 45 0

EQUATIONS 4
**MfrCout = [700,2]
Pcoll = [28,3] !kJ/h
PColl_kW = Pcoll/3600
PColl_kWm2 = PColl_kW/(AcollAp+1e-30)
PColl_Wm2  = PColl_kWm2*1000


***********************************
** Monthly printer
***********************************

CONSTANTS 1
unitPrintSol = 31

ASSIGN temp\SOLAR_MO.Prt unitPrintSol

UNIT 32 TYPE 46
PARAMETERS 6
unitPrintSol ! 1: Logical unit number, -
-1           ! 2: Logical unit for monthly summaries, -
1            ! 3: Relative or absolute start time. 0: print at time intervals␣
↪relative to the simulation start time. 1: print at absolute time intervals. No␣
↪effect for monthly integrations
-1           ! 4: Printing & integrating interval, h. -1 for monthly integration
1            ! 5: Number of inputs to avoid integration, -
1            ! 6: Output number to avoid integration
INPUTS 4
Time  Pcoll_kW  PColl_kWm2  IT_Coll_kWm2
**
Time  Pcoll_kW  PColl_kWm2  IT_Coll_kWm2

***********************************
** Hourly printer
***********************************

CONSTANTS 1
unitHourlyCol = 33

ASSIGN    temp\SOLAR_HR.Prt    unitHourlyCol
```

(continues on next page)

(continued from previous page)

```
UNIT 34 TYPE 46     ! Printegrator Monthly Values for System
PARAMETERS 7
unitHourlyCol ! 1: Logical unit number, -
-1            ! 2: Logical unit for monthly summaries, -
1             ! 3: Relative or absolute start time. 0: print at time intervals␣
↪relative to the simulation start time. 1: print at absolute time intervals. No␣
↪effect for monthly integrations
1             ! 4: Printing & integrating interval, h. -1 for monthly integration
2             ! 5: Number of inputs to avoid integration, -
4             ! 6: Output number to avoid integration
5             ! 7: Output number to avoid integration
INPUTS 6
Pcoll_kW  PColl_kWm2  IT_Coll_kWm2 TCollOut TCollIn MfrColl
**
Pcoll_kW  PColl_kWm2  IT_Coll_kWm2 TCollOut TCollIn MfrColl
```

A specific parametrization can be added by using a ddck from the database for example the type1_CONSTANTS_cOBRAak2_8v.ddck:

```
******************************
**BEGIN Type1_Constants_CobraAK2_8V.ddck
******************************

****************************************
** Solar Thermal Data for covered collector.
** Very well performing collector Cobra AK 2.8V
** Version : v0.0
** Last Changes: Jeremias Schmidli
** Date: 10.03.2020
****************************************

CONSTANTS 11

Eta0= 0.857     ! Eta0 (a0) of collector (zero heat loss efficiency)
a1 = 4.16*3.6    ! linear heat loss coefficient of collector [kJ/hm^2K] ![W/m2K]*3.6
a2 = 0.0089*3.6   ! quadratic heat loss coefficient of collector [kJ/hm^2K^2] ![W/
↪m2K2]*3.6

AbsorberArea = 2.435 !m2
TotArea = 2.768 !m2

nSeries = 1
efficiencyMode = 1
testedMfr = 200/AbsorberArea !l/hm2

GroundReflectance = 0.2

FirstOrderIAM = 0.108
SecondOrderIAM = 0
******************************
**END Type1_Constants_Test.ddck
******************************
```

Placeholder statements with specific syntax can be added to the `inputs from hydraulic solver` and `Outputs to the hydraulic solver` in ddck:

```
*******************************
**BEGIN Type1.ddck
*******************************

****************************************
** Contact person : Dani Carbonell
** Creation date  : 10.01.2010
** Last changes   : 18.05.2022
****************************************

************************************************************************
** Description:
** Collector model using efficiency curve efficiency
************************************************************************

**********************************
** inputs from hydraulic solver
**********************************
EQUATIONS 2
TCollIn = @temp(In, TPiColIn)
MfrColl = ABS(@mfr(In, MfrPiColIn))

**********************************
** outputs to hydraulic solver
**********************************
EQUATIONS 1
@temp(Out, TCollOut) = [28,1]

**********************************
** outputs to other ddck
**********************************

*********************************************************************************************
** outputs to energy balance in kWh and ABSOLUTE value
** Following this naming standard :
** qSysIn_name, qSysOut_name, elSysIn_name, elSysOut_name
*********************************************************************************************
EQUATIONS 1
qSysIn_Collector = PColl_kW

**********************************
** Dependencies with other ddck
**********************************
EQUATIONS 1
pumpColOn = puColOn

CONSTANTS 2
C_tilt = slopeSurfUser_1            ! @dependencyDdck Collector tilt angle / slope [°]
C_azim = aziSurfUSer_1              ! @dependencyDdck Collector azimuth  (0:s, 90:w,␣
↪270: e) [°]

EQUATIONS 4
**surface-8
IT_Coll_kJhm2 = IT_surfUser_1               ! Incident total radiation on collector␣
↪plane, kJ/hm2
IB_Coll_kJhm2 = IB_surfUser_1               ! incident beam radiation on collector␣
↪plane, kJ/hm2
```

(continues on next page)

(continued from previous page)

```
ID_Coll_kJhm2 = ID_surfUser_1               ! diffuse and ground reflected irradiance␣
↪on collector tilt
AI_Coll = AI_surfUser_1                     ! incident angle on collector plane, °

EQUATIONS 5
IT_Coll_kW = IT_Coll_kJhm2/3600             ! Incident total radiation on collector␣
↪plane, kW/m2
IB_Coll_kW = IB_Coll_kJhm2/3600     ! incident beam radiation on collector plane, kW/
↪m2
ID_Coll_kW = ID_Coll_kJhm2/3600     ! diffuse and ground reflected irradiance on␣
↪collector tilt (kW/m2)
IT_Coll_Wm2 = IT_surfUser_1/3.6
IT_Coll_kWm2 = IT_surfUser_1/3600

**********************************
** Begin CONSTANTS
**********************************
CONSTANTS 3
MfrCPriSpec = 15            ! Coll. Prim. loop spec. mass flow [kg/hm2]
AcollAp = 5                 ! Collector area
MfrCPriNom = MfrCPriSpec*AcollAp

**********************************
** Begin TYPE
**********************************
UNIT 28 TYPE 1
PARAMETERS 11
nSeries                     ! number in series
AcollAp                     ! collector area
cpBri                       ! fluid specific heat kj(kgK
efficiencyMode              ! efficiency mode
testedMfr                   ! tested flow rate kg/(hm2)
Eta0                        ! intercept efficiency
a1                          ! efficiency slope kJ/hm^2K
a2                          ! efficiency curvature kJ/hm^2K^2
2                           ! optical mode
FirstOrderIAM               ! 1st order IAM
SecondOrderIAM              ! 2nd order IAM
INPUTS 9
TCollIn
MfrColl
Tamb
IT_Coll_kJhm2
IT_H
ID_Coll_kJhm2
0,0
AI_Coll             !Flo check              ! JS: This was defined wrong before (C_
↪azim, even though it is incident angle input). Now it should be correct.
C_tilt              !Flo check              ! JS: This should be correct
*** INITIAL INPUT VALUES
20 0 10 0 0 0 GroundReflectance 45 0

EQUATIONS 4
**MfrCout = [700,2]
Pcoll = [28,3]              !kJ/h
PColl_kW = Pcoll/3600
PColl_kWm2 = PColl_kW/(AcollAp+1e-30)
```

(continues on next page)

(continued from previous page)

```
PColl_Wm2  = PColl_kWm2*1000

**********************************
** Monthly printer
**********************************
CONSTANTS 1
unitPrintSol = 31

ASSIGN temp\SOLAR_MO.Prt unitPrintSol

UNIT 32 TYPE 46
PARAMETERS 6
unitPrintSol                ! 1: Logical unit number, -
-1                          ! 2: Logical unit for monthly summaries, -
1                           ! 3: Relative or absolute start time. 0: print at time␣
↪intervals relative to the simulation start time. 1: print at absolute time␣
↪intervals. No effect for monthly integrations
-1                          ! 4: Printing & integrating interval, h. -1 for monthly␣
↪integration
1                           ! 5: Number of inputs to avoid integration, -
1                           ! 6: Output number to avoid integration
INPUTS 4
Time  Pcoll_kW  PColl_kWm2  IT_Coll_kWm2
**
Time  Pcoll_kW  PColl_kWm2  IT_Coll_kWm2

**********************************
** Hourly printer
**********************************
CONSTANTS 1
unitHourlyCol = 33

ASSIGN    temp\SOLAR_HR.Prt    unitHourlyCol

UNIT 34 TYPE 46             ! Printegrator Monthly Values for System
PARAMETERS 7
unitHourlyCol               ! 1: Logical unit number, -
-1                          ! 2: Logical unit for monthly summaries, -
1                           ! 3: Relative or absolute start time. 0: print at time␣
↪intervals relative to the simulation start time. 1: print at absolute time␣
↪intervals. No effect for monthly integrations
1                           ! 4: Printing & integrating interval, h. -1 for monthly␣
↪integration
2                           ! 5: Number of inputs to avoid integration, -
4                           ! 6: Output number to avoid integration
5                           ! 7: Output number to avoid integration
INPUTS 6
Pcoll_kW  PColl_kWm2  IT_Coll_kWm2 TCollOut TCollIn MfrColl
**
Pcoll_kW  PColl_kWm2  IT_Coll_kWm2 TCollOut TCollIn MfrColl

**********************************
** Online Plotter
**********************************
UNIT 103 TYPE 65            !Changed automatically
PARAMETERS 12
4                                  ! 1: Nb. of left-axis variables
```

(continues on next page)

(continued from previous page)

```
2                                    ! 2: Nb. of right-axis variables
0                                    ! 3: Left axis minimum
10                                   ! 4: Left axis maximum
0                                    ! 5: Right axis minimum
100                                  ! 6: Right axis maximum
nPlotsPerSim                 ! 7: Number of plots per simulation
12                                   ! 8: X-axis gridpoints
1                                    ! 9: Shut off Online w/o removing
-1                                   ! 10: Logical unit for output file
0                                    ! 11: Output file units
0                                    ! 12: Output file delimiter
INPUTS 6
Pcoll_kW  PColl_kWm2  IT_Coll_kWm2  MfrColl
TCollOut TCollIn
Pcoll_kW  PColl_kWm2  IT_Coll_kWm2  MfrColl
TCollOut TCollIn
LABELS  3
Power_and_Mfr
Temperatures
Collector

******************************
**END Type1.ddck
******************************
```

Acknowledgments

A first version of this package was created in 2013 and since then it has evolved considerably. We would like to thank the Swiss Federal Office Of Energy (SFOE) who supported many projects related to simulations of renewable energy systems where this code has been developed. We would also like to thank the European Union's Horizon 2020 research and innovation programme for the funding received in TRI-HP under the Grant Agreement No. 81488 and in PLURAL under the Grant Agreement No. 958218. These projects allowed to dedicate efforts in sharing the code with the consortium and to make it usable for others.