

	<p>E-DYCE_D3.1_Dynamic simulation platform_28.02.2022_Final</p> <p>Dissemination Level: PU</p> <p>H2020-LC-SC3-2018-2019-2020 / H2020-LC-SC3-EE-2019</p>	
		

Project no.: 893945

Project full title: Energy flexible DYnamic building CErtification

Project Acronym: E-DYCE

Deliverable number:	D3.1
Deliverable title:	Dynamic simulation platform
Work package:	WP3
Due date of deliverable:	M18
Actual submission date:	M18 - 28/02/2022
Start date of project:	01/09/2020
Duration:	36 months
Reviewer(s):	Michal Zbigniew Pomianowski (AAU)
Author/editor:	Giacomo Chiesa (POLITO)
Contributing partners:	Politecnico di Torino (POLITO); ESTIA SA (ESTIA)

Dissemination level of this deliverable	PU
Nature of deliverable	Other

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 893945. Any results of this project reflects only this consortium's view and the European Commission is not responsible for any use that may be made of the information it contains.

Further information is available at www.edyce.eu.

Document history

Version no.	Date	Authors	Changes
0.1	06/12/2021	Giacomo Chiesa	Initial deliverable organisation draft
0.2	28/01/2022	Giacomo Chiesa, Francesca Fasano, Paolo Grasso	Draft version shared on TEAMS to E-DYCE members for suggestions
0.3	07/02/2022	Michal Pomianowski	internal review report
0.4	14/02/2022	Giacomo Chiesa	Review-corrected version
0.9	25/02/2022	Giacomo Chiesa, Evangelos Belias, Flourentzos Flourentzou	Integration of section 3.1 and Annex A by ESTIA
1.0	28/02/2022	Anne Rommerdahl Bock	Final check and submission to EC

Contributors

Partner no.	Partner short name	Name of the Contributor	E-mail
2	POLITO	Giacomo Chiesa	giacomo.chiesa@polito.it
2	POLITO	Paolo Grasso	paolo.grasso@polito.it
2	POLITO	Francesca Fasano	francesca.fasano@polito.it
5	ESTIA	Evangelos Belias Flourentzos Flourentzou	evangelos.belias@epfl.ch flourentzou@estia.ch

Table of Contents

1	Executive Summary.....	5
2	Deliverable objective and structure.....	6
3	General structure of E-DYCE simulation platform(s)	7
3.1	Investigating the applicability of DIAL+ in the E-Dyce framework.....	9
3.1.1	Introduction	9
3.1.2	Methods.....	9
3.1.3	Results and conclusions	11
3.1.4	Conclusions	13
4	The PREDYCE tool – ‘DYCE’ development action	14
4.1	PREDYCE general description.....	15
4.2	PREDYCE modules	18
4.2.1	IDF editor	19
4.2.2	Runner.....	19
4.2.3	KPIs calculator	21
4.3	Current actions.....	22
4.3.1	IDF actions.....	22
4.3.2	KPIs.....	27
4.4	Scenarios	38
4.4.1	Sensitivity analysis	38
4.4.2	Other scenarios	40
5	Application testing	41
6	Conclusions and Outlook	45
7	Bibliography	46
8	Annex A	47

Tables of Tables

Table 1	List of the simulated scenarios.....	10
---------	--------------------------------------	----

Table of Figures

Figure 1 Sample sketch of an E-DYCE simulation platform.....	7
Figure 2 Visualisation of model's geometry in DIAL+ (left) and EnergyPlus (right) simulation models	9
Figure 3 The thermal preferences menu in DIAL+ software	11
Figure 4 Total heating demand of the models with the HVAC system	11
Figure 5 Total heating demand of the models with the HVAC system	12
Figure 6 Maximum, minimum, and average indoor air temperatures given by the EnergyPlus and DIAL+ software for the free-floating models	12
Figure 7 Percentage of free-running potential given by the EnergyPlus and DIAL+ software for the free-floating models	13
Figure 8 PREDYCE I/O	16
Figure 9 Example of input JSON file for PREDYCE	17
Figure 10 Example of PREDYCE databases content	17
Figure 11 Overview of PREDYCE modular structure	18
Figure 12 Runner module workflow and creation of aggregated output.....	20
Figure 13 Simulation-based outputs	20
Figure 14 Example of KPIs calculator module structure	21
Figure 15 Sketch of the adopted losses approach	28
Figure 16 Example of energy signature graphs.....	31
Figure 17 Example of Adaptive Comfort Model points distribution graphs	34
Figure 18 Example of carpet plots	38
Figure 19 Sensitivity analysis command line options	39
Figure 20 Sample web interface for remote run	39
Figure 21 Example of post-analysis for sensitivity analysis application to retrofit use case applied to one of the TPM residential building.....	39
Figure 22 Sample residential unit	41
Figure 23 Input JSON actions	42
Figure 24 Input JSON KPIs	42
Figure 25 Example of data_res.csv inputs part	43
Figure 26 Example of data_res.csv outputs part	43
Figure 27 Example of data_res_timeseries.csv	43
Figure 28 Energy signature best case (folder 51)	43
Figure 29 Energy signature worst case (folder 2)	44
Figure 30 PPD carpet plots for best (a) and worse (b) cases	44

1 Executive Summary

Task *T3.1 - Dynamic simulation architectures* is part of *WP3 – Simulation and Optimisation Enablers* and devoted to the developments of the dynamic simulation module (a sample E-DYCE-ready simulation platform). Deliverable *D3.1 - **Dynamic simulation platform*** includes the results of the T3.1 works defining the above-mentioned sample platform, based on the adaptation of the POLITO under-development tool PREDYCE – Python semi-Realtime Energy DYnamics and Climate Evaluation – that includes: i) input module to modify input files to run EnergyPlus simulations, ii) a running module to automatically perform simulations, iii) an output analyser module devoted to read simulation (or other sources, like monitored data) outputs and iv) allows to process them including calculations of specific KPIs. D3.1 includes the usability of a tool based on a python library and associated designs of platform able to be interrelated with the E-DYCE middleware to run simulations and retrieve outputs according to different scenarios of usage. D3.1 is strictly correlated to works in T3.2 - Free running modes, being D3.2 describing extra functionalities of the sample developed simulation platform including free-running extra actions and KPIs and detailing additional scenarios of usage of the PREDYCE tool, like performance gap, that is in line with GA descriptions. The above mentioned KPIs allow to define the KPIs families identified in the DEPC protocol – see D2.4.

The D3.1 report describes the adopted approach to define a sample E-DYCE compatible dynamic simulation platform and details the-DYCE devoted actions included in the PREDYCE tool development. Additionally, a RestApi developed to run simulations is also developed and is accessible by following instructions given in this report. The latter runs a sensitivity analysis scenario giving the needed input files and retrieving the required KPIs. E-DYCE. Specific objectives and the structure of this report is detailed in the following Section 2, while the deliverable also includes sample applications to show the potential functionalities of the sample developed simulation platform.

2 Deliverable objective and structure

In line with GA, the main objective of task T3.2 is the development of the infrastructure of the dynamic simulation platform, allowing to integrate input parameters to further run energy dynamic simulation tools. It is here presented a methodological example of a dynamic simulation platform integrating the E-DYCE approach. This sample uses the EnergyPlus engine (DOE and NREL, 2020) and the simulation platform manager PREDYCE – Python semi-Realtime Energy DYnamics and Climate Evaluation –, a new under-development Python tool by POLITO. PREDYCE implementation includes different development actions. In particular, D3.1 and D3.2 refer to the E-DYCE correlated development parts, which are referred as ‘DYCE’ development action. In particular, the ‘DYCE’ action includes base tool architecture and organisation. EnergyPlus is one of the most diffused and recognized simulation engines, nevertheless, the described in this report simulation platform approach is open to be applied to different dynamic simulation tools, including future ones, by developing simple correlated interfaces able to connect simulation engines/methodologies with the same input lists – see also E-DYCE WP2 outcomes – and the same structure to produce output results.

The tool developed by POLITO is able in managing basic EnergyPlus input files (with simplified HVAC definition) produced with different existing EnergyPlus interfaces, such as OpenStudio (NREL et al., 2021), Honeybee (Roudsari, 2013), an OpenStudio-based Grasshopper interface, DesignBuilder (DesignBuilder Software, 2020), and potentially others to allow neutral application scenarios. Graphical interfaces are not currently integrated in the loop, while the EnergyPlus version supported by the developed library is v8.x (tested on v8.9) assuming the correlated IDD file. The dynamic EnergyPlus integration module is written in Python language supporting the following issues:

- Manage an input module to compile/modify EnergyPlus input data files (IDF) to run simulations by integrating existing and new wheels and interfaces.
- Define a running module to automatically perform simulations.
- Define an output module to read simulation outputs, store data and process them to: i.) retrieve feed-back information for new input compiling; ii.) generate data for E-DYCE WP4 integration; iii.) calculate needed KPIs; iv.) include incertitude in simulations by defining variation domains of specific variable, supporting sensitivity analyses.

The deliverable is organized as follows. Firstly, the general structure of the E-DYCE simulation platform is shortly described – see Section 3. Secondly, the under-development tool PREDYCE is introduced and the description of E-DYCE main correlated functionalities is given detailing the developed platform application – see Section 4. Finally, sample application scenarios are shortly detailed to illustrate potential usages of the ‘DYCE’-driven tool functionalities – see Section 5. It is important to note that the dynamic platform here developed is a sample application (in beta-version), leaving open the future possibility to develop additional platforms based on different simulation engines. The E-DYCE DEPC protocol and the basic methodological approach used for the dynamic energy platform is neutral and replicable.

NOTE: Task 3.1 is strictly correlated with Task T3.2 about the free-running modes and, therefore, main free-running E-DYCE correlated functionalities of PREDYCE are described in E-DYCE D3.2 together with specific sample applications.

3 General structure of E-DYCE simulation platform(s)

It is well known that energy simulation is an essential support to building correlated actions and is connected to the whole life of a building, from preliminary design till building operation and end of life. Particularly, hourly dynamic simulations performed both at early design and operational stages allow to support building energy performance correlated choices and/or evaluations and can lead to the definition of a new technology-neutral dynamic labelling (DEPC), including valorisation of the free-running potential of buildings. In line with the E-DYCE rationale reported in **D1.2** (EDYCE, 2021), the E-DYCE DEPC approach is based on a three-stage process supporting a first stage devoted to *'inputs and data collection'* – see WP2 deliverables – followed by a second stage focussed on *'monitoring and implementation of dynamic certification methodology'* – see D2.4 and WP3 deliverables about dynamic simulations. Finally, the third step includes *'user feedback and actuation'* including renovation roadmaps. This deliverable (D3.1) is mainly correlated to the second step, being devoted to introduce the developed approach to manage dynamic simulation tools and to calculate from given inputs required KPIs (Key Performance Indicators).

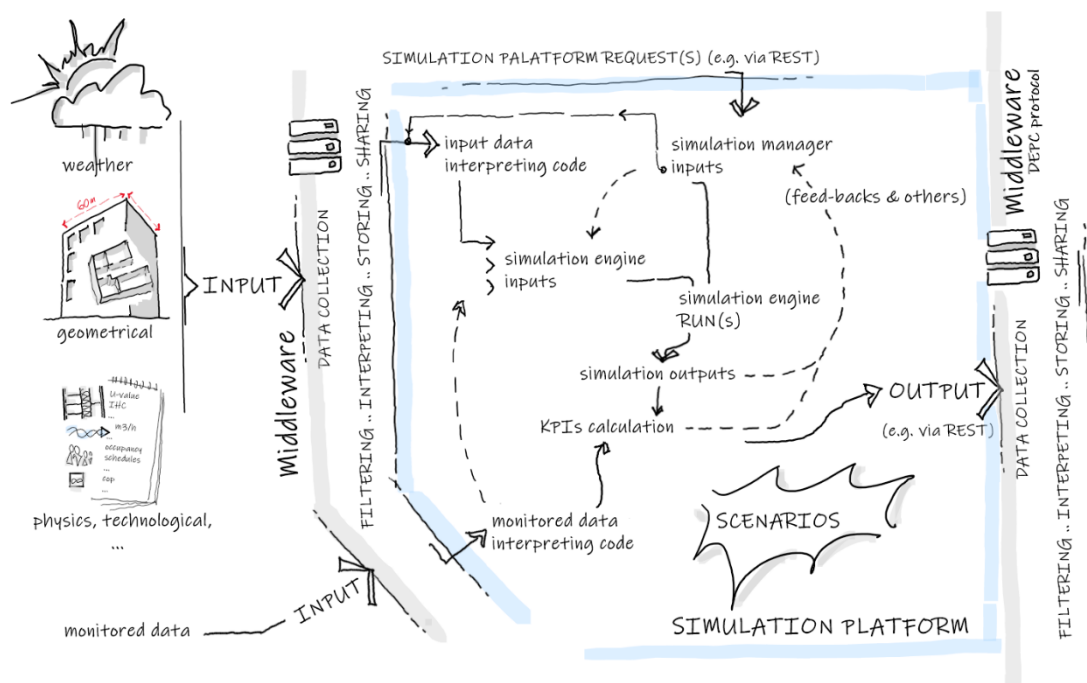


Figure 1 Sample sketch of an E-DYCE simulation platform

The general structure of an E-DYCE simulation platform is sketched in Figure 1. Assuming a series of minimal inputs needed to run dynamic simulations – see also E-DYCE WP2 results –, it is possible to develop a coding-interpreting interface connecting project stored inputs with the selected simulation engine to perform model simulation runs and retrieve simulation results. Inputs are expected to include, on the one side, geometrical data and, on the other side, building physics, technological, and operational aspects correlated to the given building – or thermal zone(s) – to perform building simulations (e.g., set points, HVAC scheduling, activities, envelope thermal characteristics, ...). Additionally, weather data – typical or monitored (historical, real-time, forecasted) – need to be also collected for simulation purposes. Input data may refer to different backgrounds, including standard conditions – e.g., profiles and data from EN 16798-1:2019 –, standard-adapted conditions – including national or regional adaptations and/or inputs from the E-DYCE inspection plan (see E-DYCE D2.2)–, and/or operational conditions. When a

request is sent to the simulation platform, including basic and interpreted simulation inputs, the latter will run, according to simulation platform running scenario, the simulation engine will collect simulation outputs. Outputs will be elaborated for KPIs definition and, according to the scenario (e.g., sensitivity analyses, simulation/monitored data comparison, etc.), results will be sent in a structured form to the enquiring actor (e.g., another E-DYCE platform agent).

According to the specific adopted simulation engine, inputs need to be elaborated and interpreted to feed simulation running requests. For example, EnergyPlus, which is the reference engine adopted in the developed simulation platform, requires two main input files: an IDF that includes structured information regarding geometries and all needed information about simulation (input data and required outputs), and a weather file in EPW format. In this example, it is possible to consider producing a basic input data file (IDF) to simulate a building using an existing EnergyPlus CAD interface allowing the simulation platform to modify this file further automatically via devoted coding definitions able to interpret specific inputs and/or input change requests. Similarly, the weather simulation file needs to be produced according to the specific simulation engine requirements using as initial inputs data from the identified weather data source. Hence, the simulation platform needs to interpret specific inputs and simulation requests to manage them supporting the generation of simulation outputs to be further elaborated into KPIs for sending back results to requiring users. Potential managing scenarios may be defined to automatically orient the simulation platform to perform for example sensitivity analyses when changing inputs (e.g., listed, range-varying, or random values) or to elaborate a comparison under real weather data of building monitored and simulated conditions expressed via KPIs.

Platform functional requirements are hence neutral in respect to the assumed simulation engine, while different simulation platforms (or internal simulation managing input/output modules) may be developed to support the calculation of needed results when different engines are adopted. In line with works performed in E-DYCE WP2, it is possible to define a list of inputs to be retrieved, e.g., by the E-DYCE inspection plan, able to feed simulation models for different existing tools, such as EnergyPlus via DesignBuilder or OpenStudio, Dial+, Termolog Dinamic module, etc. Roughly speaking, it is possible, when required KPIs calculation methods are defined – see also D2.4 and D1.2 –, to develop specific codes (e.g., python libraries) able to interpret different simulation engine inputs and to modify them according to simulation platform requests to perform simulation scenarios and retrieve comparable results to further feed E-DYCE functionalities by different dynamic simulation engines. Similarly, a potential software-house currently selling energy labelling software, may decide to include both E-DYCE simulation platform requirements and E-DYCE middleware requirements in the same tool to develop a comparable, even if specific, software and eventually choose what to include among different E-DYCE scenarios and services.

To demonstrate E-DYCE simulation functionalities, a sample simulation platform is developed, based on the above mentioned PREDYCE tool under-development by POLITO, adopting EnergyPlus as the dynamic simulation engine. Nevertheless, in Section 3.1 a comparison is provided between EnergyPlus and DIAL+, i.e., another dynamic simulation tool compatible with ISO and EN Standards, in order to demonstrate that the proposed approach is replicable in terms of results. Furthermore, even if in this deliverable a specific dynamic simulation platform is developed, the possibility to adopt different tools remain open allowing in future to adapt or develop other simulation platforms compatible with the E-DYCE middleware and PDEC protocol.

3.1 Investigating the applicability of DIAL+ in the E-Dyce framework¹

3.1.1 Introduction

Developed by Estia SA, DIAL+ is a building energy modeling software that is able to simulate daylight, natural ventilation airflows, and thermal behavior dynamically. Its ability to calculate all these parameters accurately was validated with the standards ISO 13791, EN 15255, and EN 15265.

In the framework of the E-DYCE project, it was investigated if the results produced by the DIAL+ simulations are compared with the other used simulation tool (EnergyPlus) to verify the coherence between the used tools. It is accepted in the literature that the results coming from different tools may differ, and sometimes significantly (Vadiee et al., 2019). To investigate if these two tools are comparable and if both can work for the E-Dyce project, several identical simulation models of DIAL+ and EnergyPlus were built and simulated to verify the coherence of their outputs.

3.1.2 Methods

The geometry of the models was selected to be identical with the geometry proposed by EN 15265 (2007) standard, which provides the specifications for the validation of building energy simulation tools. So, according to the above-mentioned standard, the room's internal dimensions were 3.6 x 5.5 x 2.8 m, with the window oriented towards the west. For the present analysis, it was considered that all the surfaces are in contact with the external air. A visual representation of the model geometry is presented in Fig. 2.

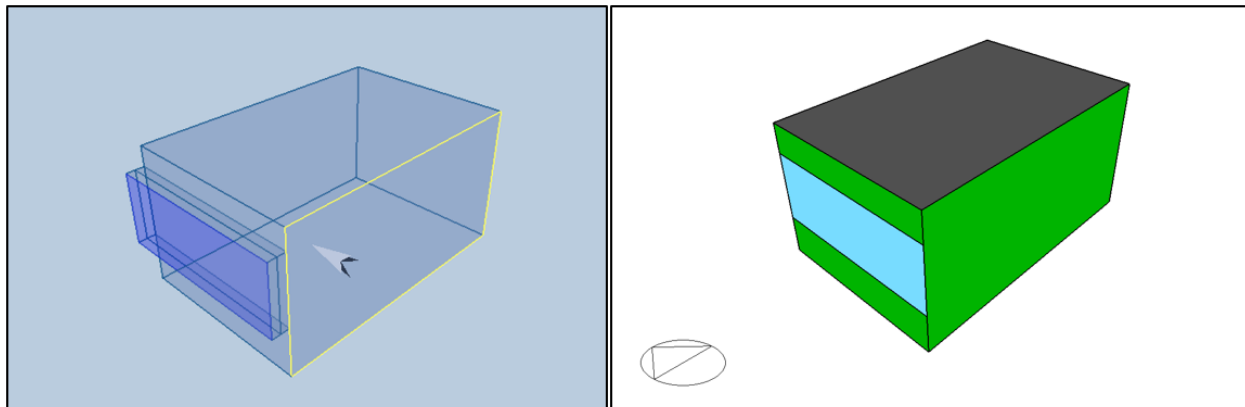


Figure 2 Visualisation of model's geometry in DIAL+ (left) and EnergyPlus (right) simulation models

To investigate how the two different software calculate the thermal loads and the indoor environmental conditions, eight different scenarios were simulated, including models with high and low thermal mass, with and without external fenestration, and with and without an HVAC system. The different simulated models with their abbreviations are presented in Table 1. The simulations were run for a typical meteorological year using the same weather file in an EPW format, as both software can read it as an input.

¹ Section 3.1 is developed by ESTIA

Table 1 List of the simulated scenarios

Thermal mass	Fenestration	HVAC system	Model's abbreviation
High thermal mass	With window	With HVAC system	H_WW_C
		Without HVAC system	H_WW_FF
	Without window	With HVAC system	H_NW_C
		Without HVAC system	H_NW_FF
Low thermal mass	With window	With HVAC system	L_WW_C
		Without HVAC system	L_WW_FF
	Without window	With HVAC system	L_NW_C
		Without HVAC system	L_NW_FF

The walls, roof, and floor of the high thermal mass model were constructed from a 15 cm concrete layer with the insulation located in the outside layer. In the lightweight model, the walls, roof, and floor had a thin layer of plaster/cement followed by a layer of insulation. In the models with fenestration, the window had a total area of 5.04 m² ($A_w = 5.04 \text{ m}^2$), and it was west-oriented. The total frame area was equal to 1 m² ($A_f = 1 \text{ m}^2$), and the glass area was equal to 4.04 m² ($A_g = 4.04 \text{ m}^2$). The U-Factor of the window (both for glass and frame) was equal to 1.1 W/m²K, with a heat gain coefficient equal to 0.4 (-).

Given that the two software have different calculation engines and use different calculation equations, a tuning of the DIAL+ thermal parameters was necessary in order for the results to be coherent enough. More specifically, the convection coefficients of outdoor surfaces were necessary to be modified to fit the specific weather file. This process was not necessary for EnergyPlus as these coefficients are auto-calculated. In addition, the outdoor coefficients of absorption of walls and roof were modified in order for the DIAL+ surfaces to match with the surfaces introduced in EnergyPlus. Lastly, for this tuning process, a modification of the correction coefficients of the heat density flow between the outside surfaces and the sky was needed in order to calibrate the models. These modifications can be done by using the "Project/Thermal preferences" tab in the menu of DIAL+, as presented in Figure 3.

The selection of the HVAC system was made accordingly in the models where the space was conditioned. The ideal air load HVAC system was selected for the EnergyPlus, and a fan coil HVAC system was set for DIAL+. The purpose of this selection was to minimize possible errors that come from the definition of more complex HVAC systems (heated slabs, radiant surfaces, etc.).

The results were compared regarding both the heating/cooling loads and other KPIs of the E-Dyce method, such as the indoor temperatures and the free-running potential. More specifically, in this study, the average and minimum/maximum temperatures, as well as the free-running hours (indoor temperature between 20 and 26 °C) of the free-floating models, were compared.

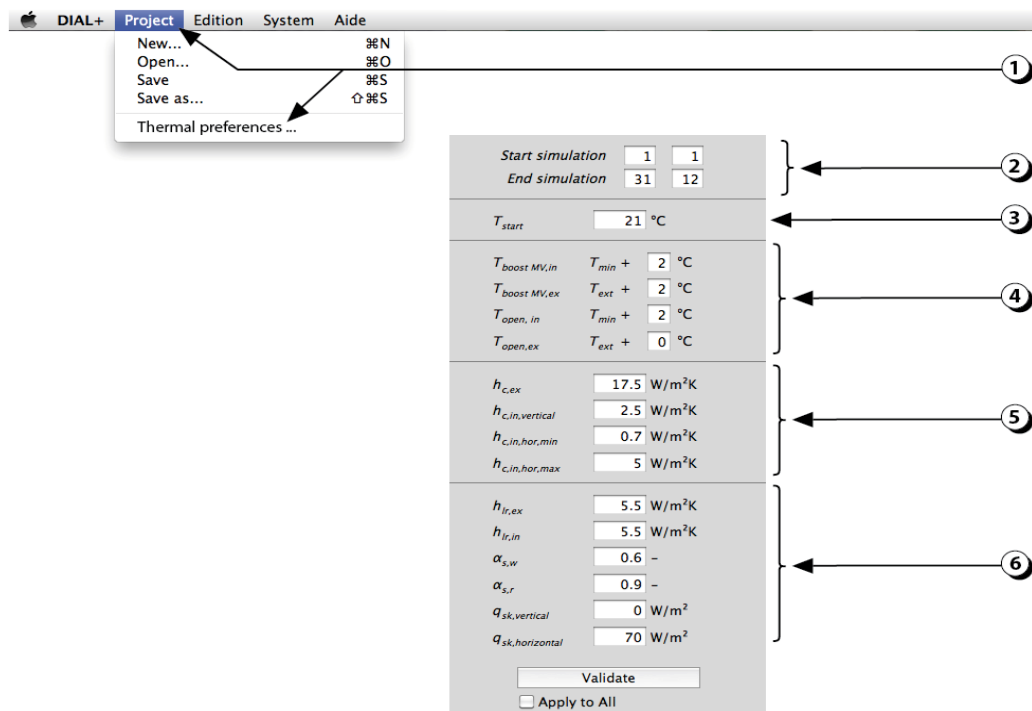


Figure 3 The thermal preferences menu in DIAL+ software

3.1.3 Results and conclusions

The results indicated that the energy need for heating given by the two software is very similar. According to the scenario, the difference is between 3% and 9%. As presented in Figure 4, when the model didn't have a window, the differences between the two software are less significant compared to the models with the window, where the differences are close to 9%.

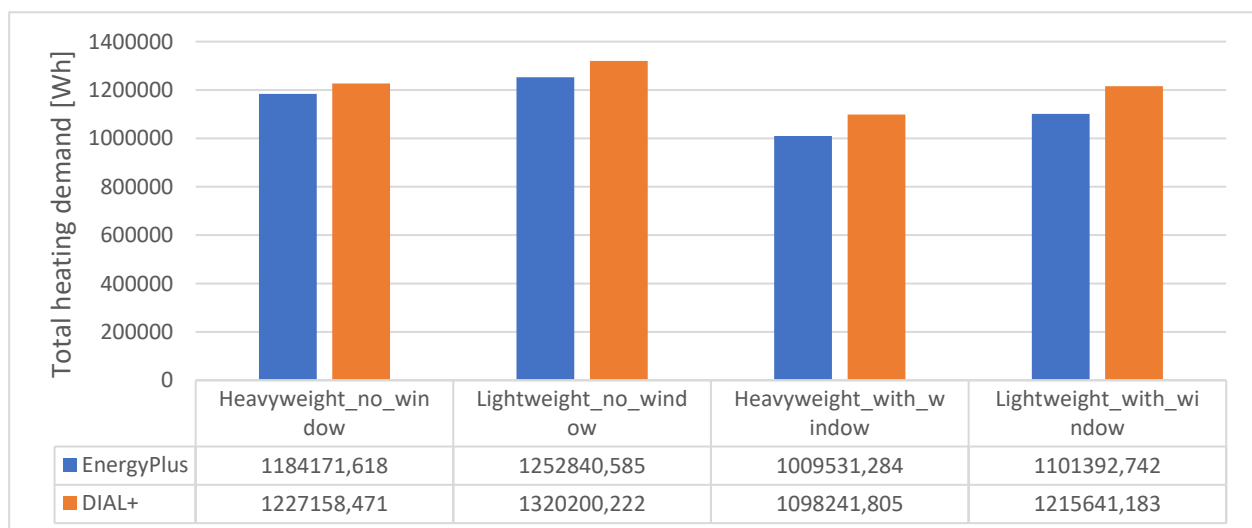


Figure 4 Total heating demand of the models with the HVAC system

As presented in Figure 5, the energy demand for cooling given by the two software is very similar. The differences are between 0% and 7.8%. Again the highest difference was observed in the model that had a window.

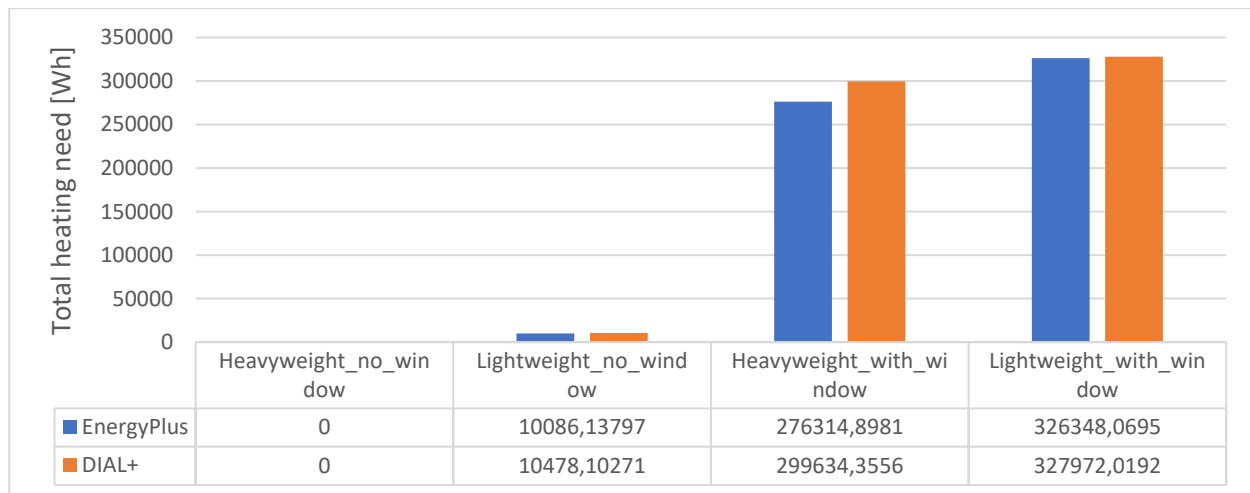


Figure 5 Total heating demand of the models with the HVAC system

Regarding the indoor temperatures generated by the two software for the free-floating models, the differences were equally not of high significance, as they are presented in Figure 6. These differences varied between 0.03°C and 0.72°C, with the highest difference being in the minimum temperature of the lightweight model with the window.

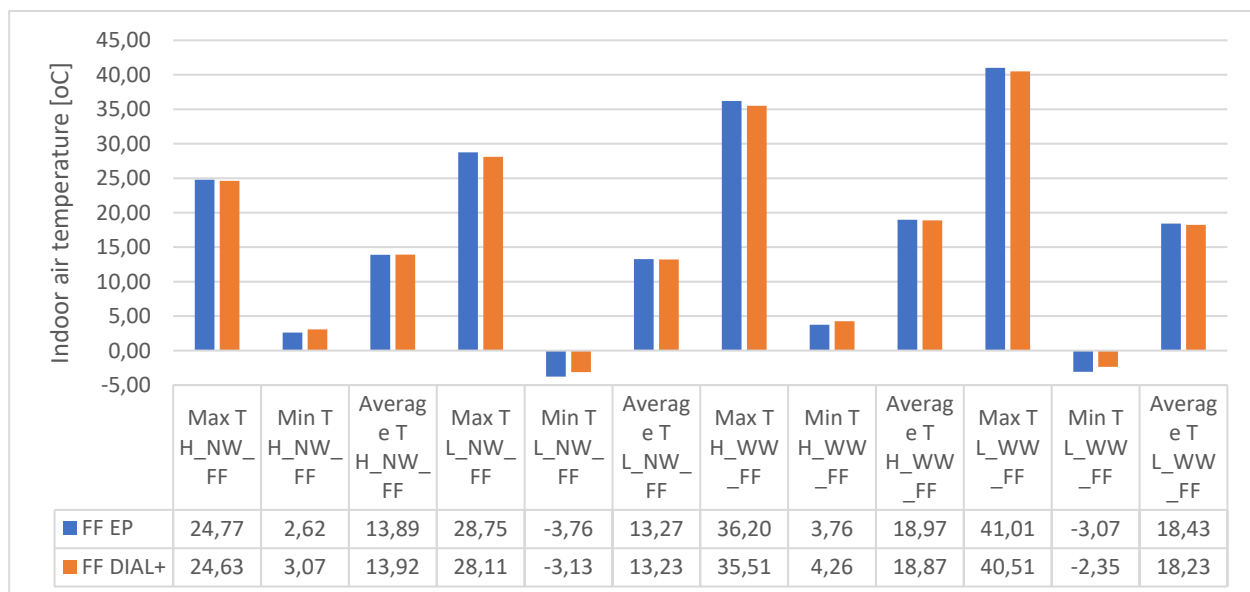


Figure 6 Maximum, minimum, and average indoor air temperatures given by the EnergyPlus and DIAL+ software for the free-floating models

Furthermore, as one of the major KPIs for the E-Dyce methods is the free-running potential of the buildings, this last was also calculated for the free-floating models. The results presented in Figure 7 indicated that the differences between EnergyPlus and DIAL+ were consistently below 10%.

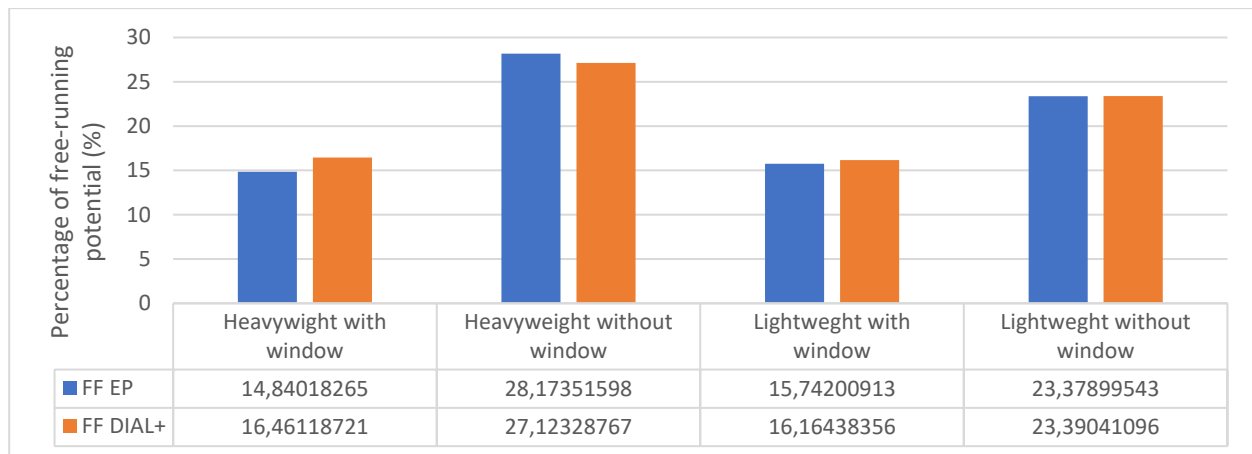


Figure 7 Percentage of free-running potential given by the EnergyPlus and DIAL+ software for the free-floating models

In the ANNEX are presented more detailed results of all the simulations run.

3.1.4 Conclusions

This study examined if the DIAL+ software can be used in the E-Dyce methodology and if the results it generates are comparable with the results delivered by EnergyPlus. The results revealed that the differences in calculating heating/cooling loads were not higher than the differences existing between all the well-known simulation tools. In addition, it can accurately calculate the free-running potential of the buildings. It is thus concluded that DIAL+ software can equally be used as a simulation platform for the simulations of the E-Dyce methodology.

Annex A includes detailed results of this analysis.

4 The PREDYCE tool – ‘DYCE’ development action

There are several available applications able to perform dynamic energy simulations for research and professional purposes, e.g., TRNSYS, Dial+, ESP-r, or IDA, each working differently concerning internal structure and computation workflow, but also input/output files format and generation. Among these energy simulation engines, EnergyPlus, which is funded by the U.S. Department of Energy (DOE), is currently one of the most diffused and recognized, at least in the academic world. Considering EnergyPlus structure and diffusion, but also its flexibility and simulation possibilities, it is adopted as dynamic simulation engine for the specific E-DYCE simulation platform here developed and proposed as sample. Particularly, inside EnergyPlus software, input data files (IDFs) are used as input to provide all information about building model geometry and activities. These files are text files based on a dictionary-like structure described in detail in the IDD file associated to each software release. Moreover, EPW files are used in input to contain weather information for an annual period (with possible variations). Both file formats have been defined together with EnergyPlus itself and despite they are quite well known they have almost no use outside the software.

EnergyPlus software can execute performance-driven energy and comfort analyses and/or optimization tasks with the help of graphical interfaces or coding tools that allow to integrate a single dynamic energy simulation in a more structured workflow including input editing and output analyses. Several graphical interfaces, periodically updated and proposing multiple functionalities, have been developed to support professionals during EnergyPlus usage; among them it is possible to mention:

- **OpenStudio:** open-source software development kit, which is also funded by DOE, containing a suite of applications allowing to create and edit building models, but also run parametric simulations and perform optimization tasks with EnergyPlus.
- **DesignBuilder:** commercial software able to perform EnergyPlus simulations including several additional professional utilities (e.g., CFD analyses). It recently allows also to perform simple parametric analyses but considering a limited number of parameters.

Graphical interfaces are currently the only working method to create an initial building model (exportable in IDF format) with complete information about all building activities (e.g., occupancy, schedules, setpoints) and complex/realistic geometry, since the structure of an IDF is quite complex and features many interconnected objects. Also, in order to actuate deep changes in building geometry (e.g., remove a wall or add a floor inside building renovation projects) in a realistic way, it is necessary to make use of these interfaces, while coding tools can be used for simpler geometry modifications.

Focusing on coding tools, the interest to develop EnergyPlus based libraries and applications to perform optimizations and other building-oriented tasks is underlined in many recent publications. Among existing projects, it is possible to mention in particular:

- **Eppy:** Python library which allows to easily edit the EnergyPlus input data files (IDFs) using a dictionary-based approach based on the input data dictionary (IDD) file provided by EnergyPlus itself; this makes the tool work with any EnergyPlus version.

- **jEPlus:** Parametric tool which allows to run multiple parallel simulations of the same building by automatically change parameters of an IDF.
- **BESOS:** Python library and JupyterHub platform which allows to create building optimization tasks by simple coding. It exploits the Eppy library to perform parametric IDF editing, and genetic or optimization algorithms (e.g., from Platypus Python library) to perform building optimization during the parametric analysis.

However, despite the numerous existing projects, a way to perform complex IDF editing tasks without the need to use a graphical interface like DesignBuilder or OpenStudio is still missing; instead, existing coding tools usually require a certain knowledge in scripting, or at least a deep knowledge of the IDF structure. Usually, this results in long, time-wasting manual operations to create a certain number of building models, making it difficult to parametrically extend the research to many building characteristics and climates. In addition, the required high-expertise level about the IDF structure makes the applicability of the mentioned tools very correlated to a deep knowledge of the EnergyPlus engine. The need of a robust simulation platform able to integrate dynamic functions, including the ones required by the E-DYCE DEPC approach, and the generation of different modifications in building specific models to calculate E-DYCE KPIs led to the development of a new tool based on readily available dynamic simulation software and open to future integration of multi-simulation engines. This new tool, on a part of which E-DYCE bases, is called PREDYCE and it is described in paragraph 4.1. Focussing on the 'DYCE' correlated development actions, its main functionalities and basic scenario are in the present deliverable D3.1, while additional E-DYCE scenarios and usages, concerning free-running buildings and comparisons between simulated and monitored data, are described in the correlated deliverable D3.2.

4.1 *PREDYCE general description*

PREDYCE (Python semi-Realtime Energy DYNAMics and Climate Evaluation) is a Python library which can be used to perform parallel runs of EnergyPlus simulations, automatic editing of IDFs (and also EPW files), and KPIs computation according to E-DYCE project requests and needs. Each module is independent from the others, such that for example KPIs calculator module can accept in input both results from simulation and structured monitored data.

The provisioned set of possible Python actions, combined with the integrated EnergyPlus launcher, can help in performing different tasks like sensitivity analyses, retrofitting suggestions or model calibration in an automatic or semi-automatic way. This combination of auto-editing and auto-running makes it possible to perform such complex tasks and output analyses without the need of writing lines of code or requiring any coding knowledge. So, taking benefits from all PREDYCE functionalities the different tasks can be organized in separate scripts which take as input and give as output files that are structured and organized in the same way. All PREDYCE scripts can be easily executed from command line, or even through a remote session with the help of a dedicated REST API, by providing the required input files as arguments. In the future each script could be treated as a pre-built scenario of usage, and automatically launched from a more general script (like an application) by just specifying the name of the scenario. Moreover, the tool can also work as a library so that expert users can exploit the already present classes and methods to create different codes with their own approach and expand the potentialities of the tool. This flexibility of usage, offered by PREDYCE modular structure, allows to give an answer to different E-DYCE needs: the organization in structured task-oriented scripts allows an easy server-to-server communication with the

FUSIX platform that acts as project middleware – see short initial description in D1.2 and works of WP4 and Task 3.3, while its organization as a library allows for high flexibility and freedom of development inside POLITO research group, making it possible to develop new scenarios of usage and testing different methodological approaches in order to further build final outputs for the project and support demo applications and tests including T4.3 (ENEA living lab) and demos' correlated tasks in WP5.

The input/output workflow of a generic PREDYCE scenario of use is described in Figure 8. At present, in order to run a PREDYCE script from command line, EnergyPlus command line launch structure made of options and an IDF model was maintained (e.g., `-w` option is followed by the EPW weather file, `-i` by the IDD version) such easing the usage of the tool for users already used to EnergyPlus software.

The main mandatory input files for any generic task are:

- the building model in IDF format,
- the weather file in EPW format,
- a JSON file, structured to contain all user requests (e.g., KPIs to be computed, parameters to be modified, run period).

While the main outputs will result in:

- a CSV file named *data_res.csv* containing aggregated KPIs in the considered run period for all the performed simulations,
- for each performed simulation, a folder containing timeseries KPIs (*data_res_timeseries.csv*) and plots is also generated.

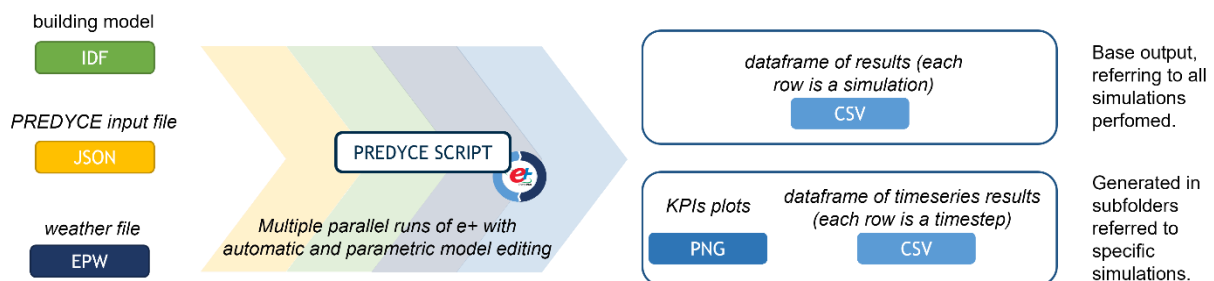


Figure 8 PREDYCE I/O

As already introduced, both input and database JSON files are used inside the tool to define actions and KPIs to be performed and retrieved during a pool of simulations and to store information necessary to modify IDFs in a dictionary form (e.g., materials composition, schedules, construction elements).

A standard PREDYCE JSON input file is structured as shown in Figure 9: the *building name* is the name of the main block of the IDF; it is utilized by the tool to know which zone elements to edit and perform calculations on. The *preliminary actions* are the actions which are executed only once before running the simulations for the parametric analyses: all simulated buildings have in common the same modifications listed in the preliminary action section. The *actions* are the parametric modifications that have to be applied to the building; all actions and their parameters are combined together to perform a set of simulations: each simulation contains a specific variation (combination of actions) of the initial building. The *kpi* section includes the final indicators that are computed at the end of each simulation. Other keys

in the input JSON file can be used for example to define a time period shorter than the run period on which to compute KPIs (to exclude the warmup period), or to define different aggregations of thermal zones on which computing the same KPIs. The *scenario* key can be used in the future to understand the requested macro task and select inside a general application which script to run.

```
{
  "building_name": "MainBlock",
  "preliminary_actions": {
    "change_runperiod": {
      "start": "01-05",
      "end": "30-09",
      "fmt": "%d-%m"}},
  "actions": {
    "change_ach": {
      "ach": [0, 2.5, 5],
      "ach_type": ["ventilation"]},
    "add_external_insulation_walls": {
      "ins_data": [
        ["extruded polystyrene panel XPS 35 kg/m3 15 mm",
         "plaster lime and gypsum 15 mm"]],
      "Thickness": [0.03],
      "Conductivity": [0.02]}},
  "kpi": {
    "Q_c": {},
    "Q_h": {}
  }
}
```

Figure 9 Example of input JSON file for PREDYCE

PREDYCE also features two internal databases, Figure 10, including default elements and descriptions. Particularly, the *database of actions* describes default behaviours of any PREDYCE method, e.g., the name of default blind element to be added when the “add exterior blind” function is called and contains text description of the function. This database is not fully implemented in the current PREDYCE version, but it is available as a concept since this approach may simplify further development of a simple GUI (graphical user interface), allowing for automatic creation of input JSON files inside the interface. Its implementation was inspired by the concept of measures in OpenStudio. The *database of objects* contains dictionary structures of basic IDF objects which are retrieved by the IDF editing functions, e.g., the characteristic of a blind element given its name. This kind of database is fundamental to any EnergyPlus interface (e.g., both DesignBuilder and OpenStudio have a database structured like this), since any IDF editing action must follow rules for object structure contained in the IDD file associated to each software release.

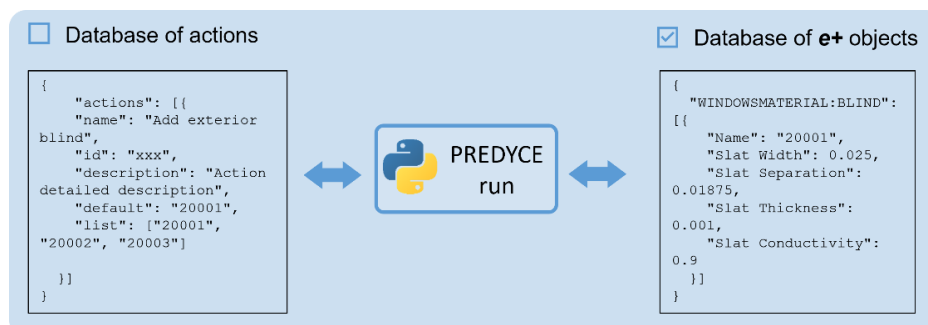


Figure 10 Example of PREDYCE databases content

4.2 PREDYCE modules

A general scheme of PREDYCE library architecture is shown in Figure 11. The tool bases on three main modules which are designed to work together but can be also used independently: i. the EnergyPlus input data file editor, which allows to act on the building model both on building structure and activities, ii. the KPIs calculator, which contains a collection of methods to compute several KPIs according to European more recent norms, and iii. the runner module, which allows to automatically manage batch of multiple parallel simulations. Moreover, additional extra-modules support other specific functionalities, e.g., to compile weather input files (EPW) from different data sources (monitored data from weather stations, forecasted data, etc.). Then, functionalities of the different modules are combined in task-oriented scripts corresponding to different scenarios of usage.

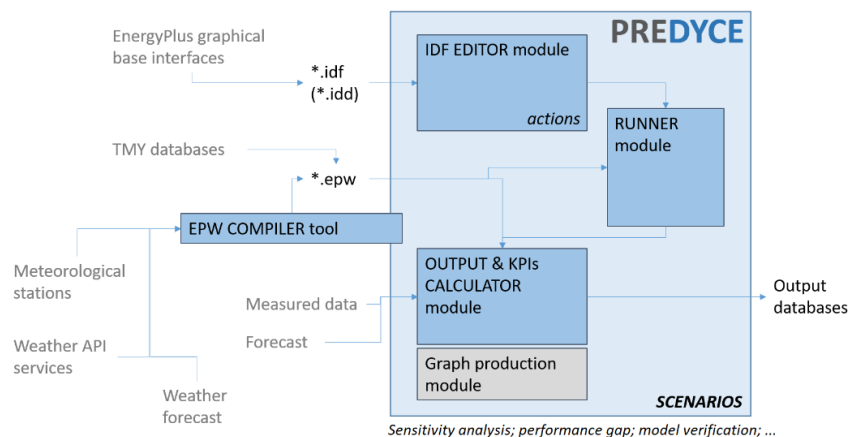


Figure 11 Overview of PREDYCE modular structure

In the following the steps of a generic scenario application workflow are reported completely, starting from building model creation to post-analysis, highlighting the automated and not automated processes:

Not automated steps

- (1) Create a base building model for EnergyPlus (IDF) through any interface able to export the IDF format, according to suggestions about model input definition defined in WP2, following standards and standard modified data after the inspection plan. Such models can then be stored inside middleware platform for reuse inside E-DYCE project.
- (2) Compile manually an input JSON file, defining actions and parameters ranges then used to modify the base of the building model. Some pre-defined JSON file to run recurrent scenarios for demos are expected to be stored directly in the project middleware.
- (3) Launch script from command line or remotely through server-to-server REST API (in this case the launch can be automatic but requires some scripting competences) or basic web interface, specifying as arguments at least the building model, the weather file, and the input JSON file.

Automated steps

- (4) Create a structured table containing permutation of parameters to be applied on the base building model: each table line corresponds to a simulation run and each column to building characteristics, Figure 12.
- (5) Run N simulations in parallel according to CPU characteristics. Rows of the previously built table contain information used for the creation of the N different IDF files. Then, after each simulation run, KPIs are computed and plots generated. Both aggregated and timeseries results are stored.
- (6) Create lightweight (limiting required storage) CSV files containing results for all simulations.

Not automated step

- (7) Perform post-analysis (e.g., finding the optimum) and plots generation of multi-simulation results.

Particularly, steps (2) and (3) have the potentialities to be further automatized through the development of a simple GUI (graphical user interface) thanks to the adopted JSON approach, but this work is not expected for the E-DYCE project supporting potential future expansions. Furthermore, step (7) could be integrated in the automated process by a future development of a post-process graph production module and introducing optimization algorithms in the loop. User interfaces and graphical restitutions are expected to be retrieved by the E-DYCE middleware and for this reason they are not part of the DYCE action in the PREDYCE tool development.

4.2.1 IDF editor

The IDF editor module contains a collection of methods able to modify EnergyPlus building models for different purposes. Such models are handled thanks to Eppy scripting language. The complexity of the actions that can be performed in order to modify the building model exclusively depends on how the different elements of the file are connected together and interpreted by EnergyPlus during a simulation run; for this reason, a manual editing of the input file can be performed only by an expert user knowing well the internal structure of the file and interconnections occurring during a simulation. The editor module instead offers a way to perform some of such complex actions by means of Python methods easily callable from a custom script, simplifying the editing procedure.

Since IDF structure depends on the EnergyPlus version release, the IDF editor methods were developed having in mind a specific software version, precisely 8.9. So, in order to be compatible with more recent software releases some functions may require a deep update and an automatic way to choose among different action versions based on the specific release should be somehow organized if there will be interest in managing this problem. This is currently one of the main limitations of the developed platform.

4.2.2 Runner

The runner module is a script including a Python class that can be instantiated and used by a user or by other PREDYCE modules to perform multiple simulations and analyses at once. The runner is in charge of creating a pool of simulations that is then executed in an asynchronous way by multiple instances of EnergyPlus on the same machine according to the number of cores of the current CPU (this can be both specified by the user or defined automatically in the process); this pool of simulations can be a simple

combination of all possible actions described in the input JSON file by the user. Also, being the proposed tool conceived to be run on both personal computers and servers this feature allows for massive simulations or the future development of services.

As described in Figure 12, each simulation is executed on a building which has been modified by one or more actions as stated in the input file. After each simulation run, all KPIs are calculated and appended in the final main output table. In the end, a final data frame of all simulated buildings is saved on a CSV file to be further analysed through cross-simulation considerations.

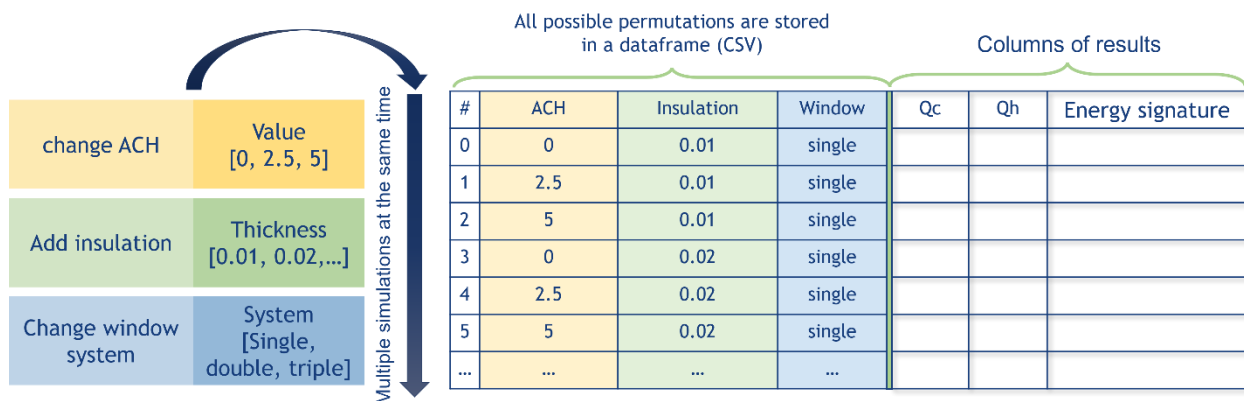


Figure 12 Runner module workflow and creation of aggregated output

Besides aggregated outputs, the runner module also creates a subfolder for each simulation containing timeseries results and plots (the match is created by naming the folder with the row index in aggregated results corresponding to the specific run), Figure 13. This is done to keep in memory more detailed information about each simulation and allowing a much lighter storage usage rather than keeping all EnergyPlus output files for further analysis. Timeseries results are by default saved with hourly resolution, but on request also other time aggregations are possible. Plots generation is instead controlled by a Boolean flag which is passed to KPI methods.

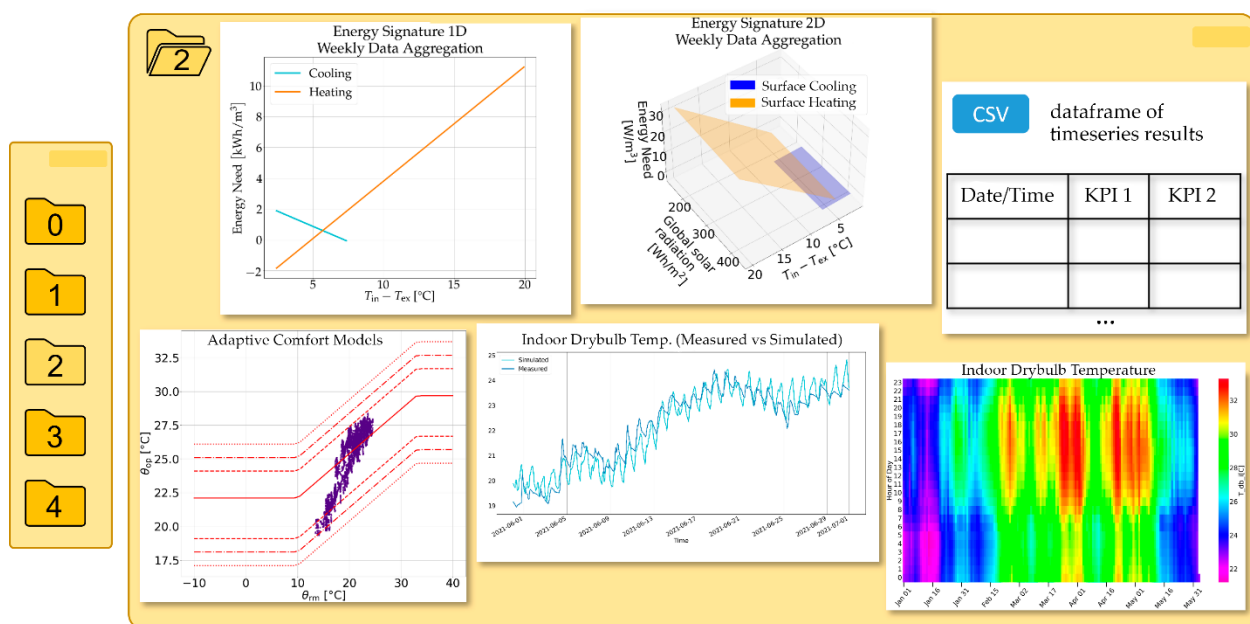


Figure 13 Simulation-based outputs

4.2.3 KPIs calculator

The KPIs calculator module is able to perform calculations, analyses and graph plots by accessing EnergyPlus output files. Such actions can be automatically performed after each simulation, called by the runner module, or manually executed by a custom Python script. The analyses usually consist in resampling output data and applying formulas to compute indicators based on European standards. Some calculations can also include graphs that can be saved in a subfolder structure to be associated to single simulations and visualized when needed. In the common case in which KPIs methods are called by the runner module, all results are aggregated together in the final CSV file in which each row represents a simulation and each column an output (including EnergyPlus base outputs if needed or a computed KPI).

Figure 14 shows how this module is built: inside KPIs calculator module there are two main classes containing the same methods, one is working on EnergyPlus outputs, while the other both on structured monitored data and on simulation results (or also from data arriving from different sources like other energy simulation tools as Dial+, if going through an intermediate step allowing for structure and nomenclature standardization). Hence, the module may treat simulated and monitored data considering both buildings and climate issues. The main goal of *EnergyPlusKPI* class is to read EnergyPlus outputs (mainly *eplusout.csv*) and prepare them (e.g., standardizing nomenclature, organizing average values on thermal zones, changing unit of measure) to actual KPI computation made by parent *KPI* class. This structure made of parent and child classes allows for very high flexibility of data sources, just requiring a common variable nomenclature, for example, as shown in Figure 14, “*T_db_o[C]*” is the variable name for the outdoor dry-bulb temperature in degree Celsius – see also **D3.2** for extra information. Among KPIs, some could return a number, while others could return a dictionary of values (e.g., describing a distribution) allowing for more complex post-analyses. Moreover, timeseries KPIs return a table with hourly associated values.

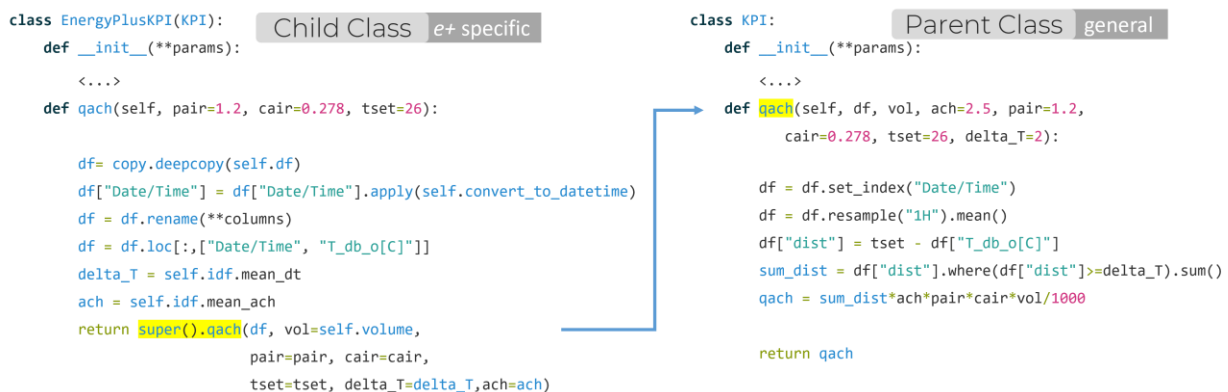


Figure 14 Example of KPIs calculator module structure

PREDYCE is organized in a way allowing to potentially integrate and expand the list of retrievable KPIs in order to be adapted to further project requests and to extra functionalities in order to allow future integrations based on results of other project WPs and tasks.

4.3 Current actions

PREDYCE is under implementation thanks to two main developing actions: i. the “DYCE” action, funded by the E-DYCE project, and ii. the “PRE” action. The “PRE” development includes a set of extra scripting actions and uses which are not described in the following . In the following paragraphs, main PREDYCE methods useful for E-DYCE objectives are listed and briefly analysed, considering both IDF editing actions and KPIs computation.

4.3.1 IDF actions

In this section some of the major IDF editing functions included in PREDYCE are listed. All actions require as input the IDF object (inherited from a Eppy class) and some parameters which can be mandatory or optional. The *filter_by* parameter, which is sometimes featured in the IDF editor actions, is useful to apply the modifications only to specific zones of the building. The actions usually feature a *void* return, meaning that the IDF object is edited by reference and not returned as a copy.

Several implemented functions regard the addition of some elements to the IDF: the only required parameters are the names of the objects that must be added which in turn are taken from the internal database.

A first set of actions includes modification to the envelope, for example the addition of insulation to the walls, which can be done internally, externally or in the cavity via different functions. The new insulating materials are passed by their names since the complete objects are included in the internal database.

predyce.idf_editor.add_external_insulation_walls(idf, ins_data=None, filter_by="", **fields)

Add external insulation layer and plaster layer on external walls. Since layers are added externally internal building area and volume are not impacted.

Parameters

- **idf** (class:predyce.IDF_class.IDF) – IDF object
- **ins_data** (list of two string elements, optional) – Construction objects list (insulation material and plaster), defaults to None
- **filter_by** (str, optional) – Filter zones by name. Can be the block name or specific zone name, defaults to ""
- **fields** – Additional fields to be set, passed as keyword arguments

Example

```
add_external_insulation_walls(
    idf,
    ins_data=[
        [
            "extruded polystyrene panel XPS 35 kg/m3 15 mm",
            "plaster lime and gypsum 15 mm",
        ]
    ],
    filter_by="kitchen",
    Conductivity=0.4,
)
```

The addition of insulation can also be performed on the floors both internally and externally, in an analogue way, and also on flat and tilted roofs, depending on the building, both internally or externally.

predyce.idf_editor.add_external_insulation_floor(*idf*, *ins=None*)

Add insulation on the external layer of all floors in the building.

Parameters

- **idf** (class:predyce.IDF_class.IDF) – IDF object
- **ins_name** (str, optional) – Name of the insulation which has to be added to the floor, defaults to None

predyce.idf_editor.add_insulation_flat_roof(*idf*, *ins_name=None*, *sheathing_name=None*)

Add insulation to flat roof, under the sheathing layer.

If no insulation is provided, a default one would be used; if no sheathing layer is provided, the function will try to put the insulation under the fibreboard if present or under the asphalt.

Parameters

- **idf** (class:predyce.IDF_class.IDF) – IDF object
- **ins_name** (str, optional) – Name of the insulation which has to be added to the roof, defaults to None
- **sheathing_name** (str, optional) – Name of the sheathing layer under which the insulation has to be added, defaults to None

There is the possibility also to substitute the entire window system by changing the glazing type, e.g., simple, double or triple glazing with different types of gas. The only requirement is that the new element is present in the internal database.

predyce.idf_editor.change_windows_system(*idf*, *win_type=None*)

Change all Windows and GlazedDoors system with a different construction object.

Parameters

- **idf** (class:predyce.IDF_class.IDF) – IDF object
- **win_type** (str, optional) – Construction name, defaults to None

Another type of action regards the modification of the window to wall ratio of the existing windows. There changing of the window-to-wall ratio (WWR) can also be made by orientation if a dictionary containing pairs of orientations and values is passed to the function.

predyce.idf_editor.change_wwr(*idf*, *wwr_value*=0.2, *wwr_map_value*={})

Change wwr value for all windows in an IDF object or specify in *wwr_map_value* dictionary wwr corresponding to specific windows orientation.

Parameters

- **idf** (class:*predyce.IDF_class.IDF*) – IDF object
- **wwr_value** (float in range [0,1]) – wwr for all windows, if not specified it is 20%
- **wwr_map_value** (dict, {int (0,90,180,270): float (in range [0,1])}) – dictionary where each key corresponds to a orientation and value to wwr.

The IDF editor can also edit the HVAC system of the building, for example it is possible to activate or deactivate the cooling system. If the building does not have an HVAC system at all, the function will automatically add all the necessary HVAC elements to the model by means of another function.

predyce.idf_editor.activate_cooling(*idf*, *cooling_schedule*, *cool_sch_name*='Cooling SP Sch', *cool_avail_name*='Cooling Availability Sch', *filter_by*=')

Activate cooling on the building. If the building does not have an HVAC system, the entire system is added.

Parameters

- **idf** (class:*predyce.IDF_class.IDF*) – IDF object
- **cooling_schedule** (dict) – New schedule for the cooling system.
- **cool_sch_name** (str, optional) – Name of the cooling setpoint schedule compact which has to be replaced, defaults to “Cooling SP Sch”
- **cool_avail_name** (str, optional) – Name of the cooling availability schedule compact which has to be replaced, defaults to “Cooling Availability Sch”
- **filter_by** (str, optional) – Filter zones by name. Can be the block name or specific zone name, defaults to “”, ignored if ach is a dict

predyce.idf_editor.add_hvac(*idf*, *heating_schedule*=None, *cooling_schedule*=None, *filter_by*=')

Add HVAC to a building without HVAC.

Parameters

- **idf** (class:*predyce.IDF_class.IDF*) – IDF object
- **heating_schedule** (dict, optional) – Heating schedule, defaults to None
- **cooling_schedule** (dict, optional) – Cooling schedule, defaults to None
- **filter_by** (str, optional) – Filter zones by name. Can be the block name or specific zone name, defaults to “”

Another set of actions include the possibility to edit specific parameters inside the model, for example it is possible to change the setpoints of heating and cooling: in this case the function will edit the heating/cooling schedules by identifying and modifying the proper values.

predyce.idf_editor.change_setpoint(*idf*, *heat=None*, *cool=None*)

Change heating and cooling setpoints in the IDF.

Parameters

- **idf** (class:*predyce.IDF_class.IDF*) – IDF object
- **heat** (*int*, *optional*) – New heating setpoint value, defaults to None
- **cool** (*int*, *optional*) – New cooling setpoint value, defaults to None

Another change can be made on the U-factor of the objects: it is possible in fact to edit the R-value of walls, pavements, ceilings, roofs and windows. In all cases the function will try to modify the R-value of the most insulating element by changing its thickness or substituting the entire system with a simplified version with the desired R-value.

predyce.idf_editor.change_ufactor(*idf*, *ufactor*, *where='Walls'*, *relative=False*)

Change Ufactor of specified Construction objects to desired value acting on thickness of most insulating layer in the construction object.

Parameters

- **idf** (class:*predyce.IDF_class.IDF*) – IDF object
- **ufactor** (*float*) – desired final ufactor value
- **where** (*string*, *defaults to "Walls"*) – where to compute Ufactor, could be “Pavements”, “Walls”, “Ceilings”, “Roof” or the name of a specific Construction object

predyce.idf_editor.change_ufactor_windows(*idf*, *value*)

Set an U-Factor value to all windows of type SimpleGlazingSystem.

Parameters

- **idf** (class:*predyce.IDF_class.IDF*) – IDF object
- **value** (*float*) – New U-Factor value

When windows are simple glazing, a modification of the solar heat gain coefficient can be performed easily.

predyce.idf_editor.change_shgc(*idf*, *value*)

Set an SHGC (Solar Heat Gain Coefficient) value to all windows of type SimpleGlazingSystem.

Parameters

- **idf** (class:*predyce.IDF_class.IDF*) – IDF object
- **value** (*float*) – New SHGC value

Some simple modifications which can be easily made by hand are also supported by the tool, for example it is possible to change the running period of the simulation or the occupancy of the building by calling a

function; regarding the occupancy, it can be set randomly by specifying the minimum and maximum values of the random interval.

predyce.idf_editor.change_runperiod(*idf, start, end, fmt='%d-%m'*)

Change simulation RunPeriod according to provided dates.

Parameters

- **idf** (class:*predyce.IDF_class.IDF*) – IDF object
- **start** (*str*) – Start date
- **end** (*str*) – End date
- **fmt** (*str, optional*) – Format of the date, defaults to “%d-%m”

predyce.idf_editor.change_occupancy(*idf, value=None, schedule=None, filter_by="", relative=False, low=0.03, high=0.2, replace=False*)

Change People per Zone Floor Area field with a new value.

Parameters

- **idf** (class:*predyce.IDF_class.IDF*) – IDF object
- **value** (*float or str, optional*) – New base occupancy value; it can be a float number or “random”. If set to “random”, minimum and maximum values for the random interval can be provided by *low* and *high* parameters, defaults to *None*
- **schedule** (*dict, optional*) – New occupancy schedule, defaults to *None*
- **filter_by** (*str, optional*) – Filter zone by name. Can be the block name or specific zone name, defaults to “”
- **relative** (*bool, optional*) – Specify if new value will be assigned as a percentage increment from the old value, defaults to *False*
- **low** (*float, optional*) – Minimum base occupancy value when “random” is chosen for *value* parameter, defaults to 0.03
- **high** (*float, optional*) – Maximum base occupancy value when “random” is chosen for *value* parameter, defaults to 0.2
- **replace** (*bool, optional*) – If *True*, the current schedule is replaced by the new one, defaults to *False*

Warning

When *replace* is set to *True*, schedule associations are preserved; therefore you might unintentionally modify several elements in the model. This happens if the modified schedule is also associated to other element than occupancy.

4.3.2 KPIs

Concerning implemented KPIs, here are reported methods belonging to the parent *KPI* class which contains the actual indicators calculation workflows, that can be applied to data originating from any data source, while the *EnergyPlusKPI* class, inheriting from *KPI*, contains EnergyPlus outputs preprocessing and then recalls parent methods. As can be seen in the following, any *KPI* method requires in input at least a data frame containing columns with standardized nomenclature (concept expanded in correlated deliverable D3.2), then other parameters that can be mandatory or optional. Consequently, the only required step for accepting data from different sources is to standardize them to the proposed table format, making the KPIs computation technology neutral and independent from adopted simulation engines.

Methods inside *KPI* class can be grouped into three macro-groups according to the output type: KPIs returning aggregated results for the requested time period in numerical or dictionary form (and eventually correlated graphs for further visual analysis); KPIs returning timeseries results, so tables with date/time and indicators; KPIs returning only graphs for further visual inspection. Moreover, indicators may be grouped in thematic groups, e.g., energy operation KPIs, energy signature of a building (or zone), comfort/quality KPIs, free-running KPIs. In the following parts of the technical manual describing some of the main KPIs implemented for E-DYCE are reported; then, they are combined inside the scenarios of usage in order to perform a more complex and complete building analysis.

Concerning consumption related KPIs, in accordance with WP2 outcomes, the main indicators are final (f_Q_x) and primary (Q_x) energy needs for heating, cooling, lighting, etc. Since EnergyPlus outputs for consumption are expressed in form of net energy needs, so before any losses occurrence, there was the need of considering losses' correction factors inside the KPIs computation. Moreover, usually also consumption monitoring may be performed at least before the occurrence of generator losses, e.g., by using heat flow meters. Consequently, hourly consumption values can be corrected with simple numeric factors provided as parameters by the final user (through the input JSON file) according to national or European norms and specific demo case characteristics, before being aggregated on the considered time period. This allows future developments, including, if needed, the implementation of more complex losses' formulas, not requiring anymore to input the correction factors. Considering demo case consumption typical monitoring, the choice of applying by default only generator losses correction on these data was taken, while emission, regulation and distribution losses correction factors are applied on simulation results. The developed KPIs are in fact adopting the UNI-TS 11300-2 approach supporting the definition of heating and domestic hot water final energies from zone net energies, by considering i.) utilisation losses, including in order i.a) emission losses, i.b) regulation losses, and i.c) distribution losses, and ii.) generation losses (see also UNI-TS 11300-4) – a sketch of these losses is reported in Figure 15. For cooling a similar approach is adopted in line with UNI-TS 11300-3. Nevertheless, the organisation of these specific KPIs allow, thanks to the multiple losses' points, to integrate different approaches, by for example skipping not needed ones (setting them equal to 1) or substituting values with expressions or databases – in line with the above-mentioned potential future expansions.

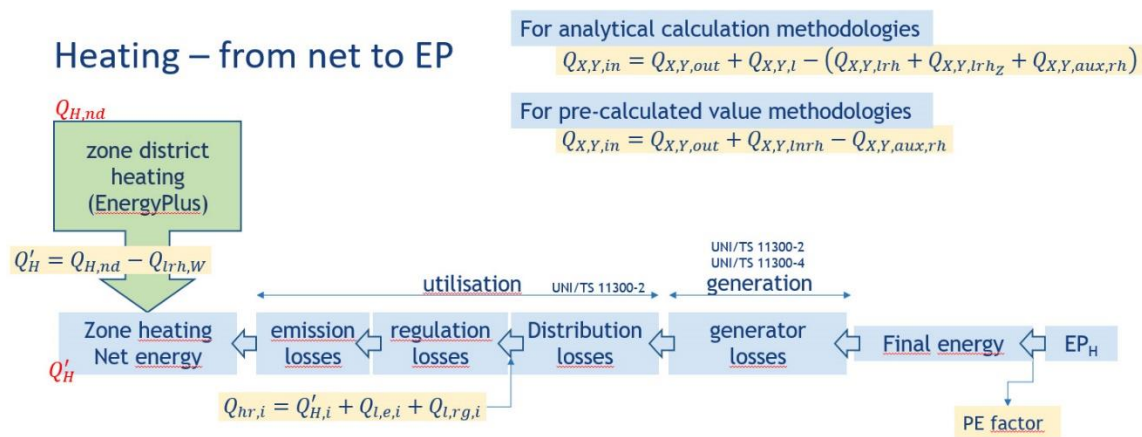


Figure 15 Sketch of the adopted losses approach

In case monitored data need correction at previous steps, it can be possible to use PREDYCE methods for a preprocessing, since they are simply provided as Python static methods (not requiring class instantiation).

```
static distribution_losses(value, distr_factor=1)
```

Apply distribution losses correction factor.

Parameters

distr_factor (float, default 1) – distribution factor, in range [0,1]

Returns

consumption value with correction applied

Return type

float

Here primary energy need for heating is reported as an example. The following method belongs to *EnergyPlusKPI* class and applies correction factors to distribution.

→ `class predyce.kpi.EnergyPlusKPI(df, idf, plot_dir, start_date=None, end_date=None, graph=False)`

`Q_h(em_factor=1, reg_factor=1, distr_factor=1, gen_factor=1, ep_factor=1)`

Prepare eplosout.csv to compute primary energy need for heating in kWh/m2, applying correction factors for emission, regulation, distribution and generation losses, then multiplying by primary energy factor.

Parameters

- **em_factor** (*float, default 1*) – correction factor for emission losses, in range [0,1].
- **reg_factor** (*float, default 1*) – correction factor for regulation losses, in range [0,1].
- **distr_factor** (*float, default 1*) – correction factor for distribution losses, in range [0,1].
- **gen_factor** (*float, default 1*) – correction factor for generation losses, in range [0,1].
- **ep_factor** (*float, default 1*) – primary energy conversion factor, in range [0,1].

Returns

Primary energy need for heating

Return type

float

Then, the same method belonging to the parent *KPI* class is reported. It applies remaining correction factor and converts to primary energy. In the following also *f_Q_h* is reported as an example of final energy need for heating: it only differs from primary energy for the last conversion step.

→ `class predyce.kpi.KPI(plot_dir, graph=False)`

`Q_h(df, gen_factor=1, ep_factor=1)`

Compute primary energy need for heating in kWh/m2, applying correction factors for generation losses, then multiplying by primary energy factor. Corrections for emission, regulation and distribution losses are considered previously applied.

Parameters

- **df** (*class:pandas.core.frame.DataFrame*) – Dataframe which must contain at “Q_h[kWh/m2]” and “Date/Time” columns.
- **gen_factor** (*float, default 1*) – correction factor for generation losses, in range [0,1].
- **ep_factor** (*float, default 1*) – primary energy conversion factor, in range [0,1].

Returns

Primary energy need for heating

Return type

float

f_Q_h(df, gen_factor=1)

Compute final energy need for heating in kWh/m2, applying correction for generation losses but considering correction factors for emission, regulation and distribution as already applied.

Parameters

- **df** (class:*pandas.core.frame.DataFrame*) – Dataframe which must contain at “Q_h[kWh/m2]” and “Date/Time” columns.
- **gen_factor** (float, default 1) – correction factor for generation losses, in range [0,1].

Returns

Primary energy need for heating

Return type

float

Another important KPI related to consumption is the energy signature. It is mainly a visual representation of the building energy behaviour, so it requires to return enough coordinates to create desired graphs. Particularly, the developed method can return data points useful in generating both one-dimensional (x-axis is the difference between indoor and outdoor dry bulb temperatures) and two-dimensional (x- and y-axes contain temperature difference and global solar radiation) energy signatures for heating and cooling net energy needs, considering a weekly average, as suggested in literature. Moreover, the method can also return energy signature graphs on request, as shown in Figure 16.

energy_signature(df)

Compute energy signature in a standardized format: dataframe used should contain “Date/Time”, “T_db_o[C]”, “T_db_i[C]”, “Rad_global_o[W/m2]” and HVAC consumptions columns, “Q_c[Wh/m3]” and “Q_h[Wh/m3]”.

Parameters

dataframe (class:*pandas.core.frame.DataFrame*) – dataframe containing at least “Date/Time”, “T_db_o[C]”, “T_db_i[C]”, “Rad_global_o[W/m2]”, “Q_h[Wh/m3]”, “Q_c[Wh/m3]” columns

Returns

Minimum points allowing to build 1D and 2D graphs of energy signature.

Return type

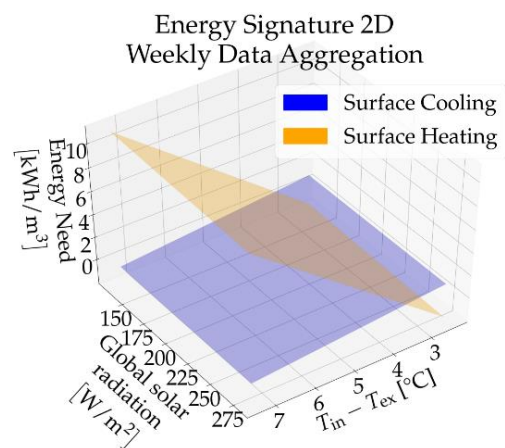
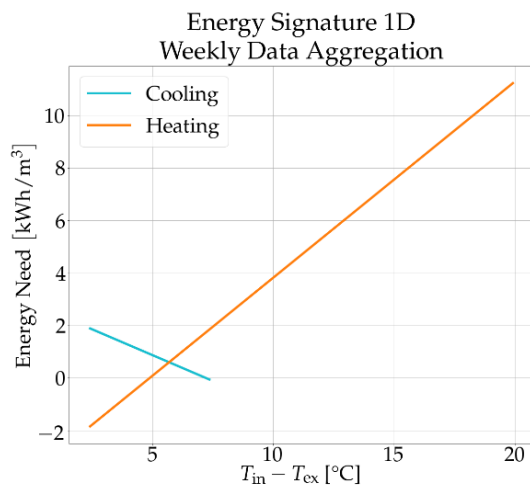
dict

Energy signature is an example of KPI returning a dictionary instead of a float, but this is the case of any more complex KPI, for example returning a distribution of points inside given categories. In the following is shown an example of energy signature output.

Example

Returned dictionary structure:

```
{
  "1D": {
    "cooling": {
      "deltaT": list of 2 points,
      "cooling": list of 2 points,
    },
    "heating": {
      "deltaT": list of 2 points,
      "heating": list of 2 points,
    },
  },
  "2D": {
    "cooling": {
      "deltaT": list of 4 points,
      "solarRadiation": list of 4 points,
      "cooling": list of 4 points,
    },
    "heating": {
      "deltaT": list of 4 points,
      "solarRadiation": list of 4 points,
      "heating": list of 4 points,
    },
  },
}
```

**Figure 16 Example of energy signature graphs**

Considering the case of building un-equipped with cooling system, as can be the case for typical and historical Mediterranean buildings, it can be useful to quantify the thermal discomfort in terms of energy consumption that could be supplied to re-establish a comfort condition. The concept behind the structure of this KPI is expanded in correlated deliverable D3.2 The same concept can be theoretically applied also to buildings un-equipped with heating system (which is a rarer case in Europe), so defining a fictitious heating indicator.

```
fict_cool(dataframe_off, dataframe_on, eu_norm='16798-1:2019', alpha=0.8)
```

Return fictitious cooling computed as the amount of cooling consumption needed in a building without mechanical system in order to reach category I comfort of adaptive comfort model.

Parameters

- **dataframe_off** (class:*pandas.core.frame.DataFrame*) – dataframe containing at least “Date/Time”, “T_op_i[C]”, “T_db_o[C]” columns in order to compute adaptive comfort model categories.
- **dataframe_on** (class:*pandas.core.frame.DataFrame*) – dataframe containing at least “Date/Time” and “Q_c[Wh/m2]” columns, of the same building with HVAC installed and ACH ventilation set to 0.
- **eu_norm** (*str, optional*) – It can be set to ‘15251:2007’ if old UE norm computation is desired, defaults to ‘16798-1:2019’.
- **alpha** (*float, optional*) – With old UE norm ‘15251:2007’ alpha is a free parameter in range [0,1), defaults to 0.8

Returns

fictitious cooling total [kWh/m2]

Return type

float

Even if the DEPC protocol mostly focusses on energy and less on comfort, PREDYCE includes the possibility to retrieve thermal comfort KPIs that will be tested in demonstration cases for potential evaluation in the final version of the protocol. Hence, concerning thermal comfort KPIs, different methods have been developed. Regarding Fanger model the calculation workflow for PMV (Predicted Mean Vote) and PPD (Predicted Percentage of Dissatisfied) was implemented in line with the one described in ISO 7730 standard, in which different comfort categories are suggested according to PMV ranges – i.e., category I (± 0.2 PMV); II (± 0.5 PMV); and III (± 0.7 PMV). Particularly, hours falling in category III are considered in thermal discomfort. The developed *pmv_ppd* function returns a dictionary containing both the number and the percentage (expressed as POR – Percentage Outside the range) of hours in category III. Default values are provided for Fanger model parameters, but they can also be set through the input JSON file: the clothing level is assumed to be 0.5 clo, while the metabolic rate is set to 1.2 met, corresponding to standing relaxed condition or sitting activities.

Moreover, Adaptive Comfort Model categories are considered to describe thermal comfort. Both old (15251:2007) and new (16798-1:2019) European standards can be used for the computation of the running mean temperature (new norm set as default procedure). Hours in category III are considered for thermal discomfort in order to compute the POR, while for a deeper analysis all categories distribution are retrieved. It is also possible to filter hours according to occupancy, such considering only occupied time periods in thermal comfort analysis. The field *when* instead can be used to reduce the analysed period with respect to simulation period. The *adaptive_residuals* function instead is used to return the average distance from categories boundaries in order to have a general look at points distribution.

Moreover, a graph can be saved showing the points distribution (with hourly aggregation) with respect to categories boundaries over the considered time period, as shown in Figure 17.

pmv_ppd(df, vel=0.1, met=1.2, clo=0.7, wme=0, filter_by_occupancy=0)

Return Predicted Mean Vote (PMV) and Predicted Percentage of Dissatisfied (PPD) calculated in accordance to ISO 7730-2006 standard.

Parameters

- **df** (class:*pandas.core.frame.DataFrame*) – dataframe containing at least “Date/Time”, “T_db_i[C]”, “T_rad_i[C]” and “RH_i[%]” columns. Optional “Occupancy column” accepting only 0 and 1 values.
- **vel** (*float, optional*) – relative air speed, defaults 0.1
- **met** (*float, optional*) – metabolic rate, [met] defaults 1.2
- **clo** (*float, optional*) – clothing insulation, [clo] defaults 0.5
- **wme** (*float, optional*) – external work, [met] defaults 0
- **standard** (*str, optional*) – default “ISO 7730-2006”, but currently unused
- **filter_by_occupancy** (*int, optional*) – It can be set 0 or 1, depending on whether activate occupancy filtering on thermal comfort KPIs computation or not, default 0.

Returns

dictionary containing keys “POR” (Percentage Outside the Range) and “No_h_discomfort”, computed as percentage and number of hours in which PMV is above or below 0.7 and so PPD is above around 20%.

Return type

dict

adaptive_residuals(dataframe_off, eu_norm='16798-1:2019', alpha=0.8)

Return average distance from Adaptive Comfort Model upper categories thresholds.

Parameters

- **dataframe_off** (class:*pandas.core.frame.DataFrame*) – Dataframe containing at least ‘Date/Time’, ‘T_db_o[C]’ and ‘T_op_i[C]’ columns.
- **eu_norm** (*str, optional*) – EU normative to compute Adaptive Comfort Model thresholds. It can be ‘16798-1:2019’ or ‘15251:2007’, defaults to ‘16798-1:2019’
- **alpha** (*float, optional*) – EU ‘15251:2007’ free parameter ranging [0,1), defaults to 0.8

Returns

Dictionary where ‘>3’: average distance from cat 1 upper bound; ‘>0’: average distance from central line of cat 1

Return type

dict

```
adaptive_comfort_model(df, eu_norm='16798-1:2019', alpha=0.8, filter_by_occupancy=0, when={})
```

Compute adaptive comfort model in a standardized format.

Parameters

- **df** (*class:pandas.core.frame.DataFrame*) – dataframe should contain “Date/Time” column in format ‘year/month/day hour:minutes:seconds’, “T_db_o[C]” preferably with a subhourly timestep and “T_op_i[C]”. Optional “Occupancy” column accepting only 0/1 values.
- **eu_norm** (*str, optional*) – It can be set to ‘15251:2007’ if old UE norm computation is desired, defaults to ‘16798-1:2019’.
- **alpha** (*float, optional*) – With old UE norm ‘15251:2007’ alpha is a free parameter in range [0,1), defaults to 0.8
- **filter_by_occupancy** (*int, optional*) – It can be set 0 or 1, depending on whether activate occupancy filtering on thermal comfort KPIs computation or not, default 0.
- **when** (*dict, optional*) – dictionary with ‘start’ and ‘end’ keys and values in format ‘year/month/day hour:minutes:seconds’

Returns

Number of hours in each of the 7 comfort categories and POR computed as % of hours outside cat 2 boundaries.

Return type

dict

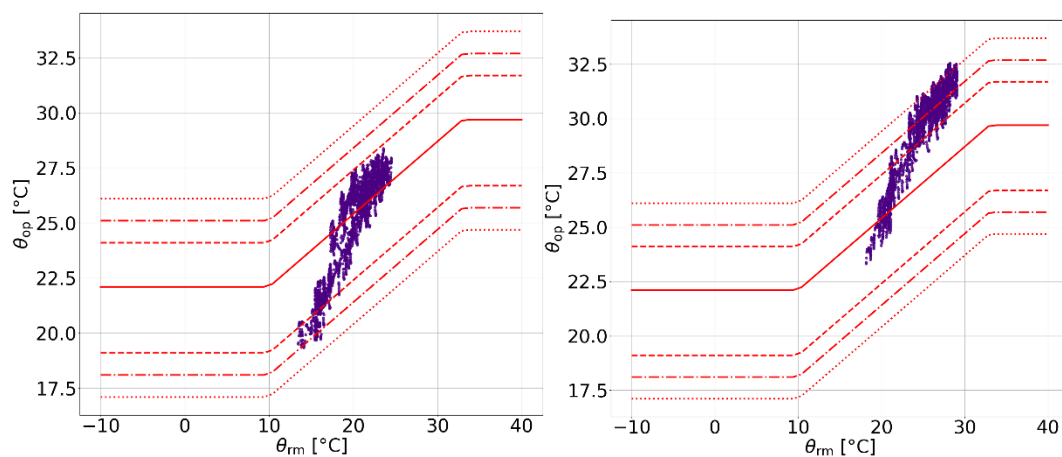


Figure 17 Example of Adaptive Comfort Model points distribution graphs

Concerning comfort-quality ‘DYCE’ related KPIs, the most important indicators underlined in WP2 outcomes regard indoor CO₂ concentration. Particularly, the number of hours below the threshold of 600 ppm and above the threshold of 1000 ppm can be retrieved, suggesting potential over- and under-

ventilated conditions. Different thresholds can be easily included if additional correlated-KPIs, e.g., confinement indices, would be necessarily during the project demonstration phase.

n_co2_all(df)

Compute number of hours with CO2 concentration above 1000 ppm highlighting possible under-ventilated conditions.

Parameters

df (class:*pandas.core.frame.DataFrame*) – Dataframe which must contain at least CO2 concentration indoor “CO2_i[ppm]” and “Date/Time” columns.

Returns

number of hours above threshold III

Return type

int

n_co2_bl(df)

Compute number of hours with CO2 concentration below 600 ppm highlighting possible over-ventilated conditions.

Parameters

df (class:*pandas.core.frame.DataFrame*) – Dataframe which must contain at least CO2 concentration indoor “CO2_i[ppm]” and “Date/Time” columns.

Returns

number of hours below threshold I

Return type

int

All groups of KPIs (energy, comfort/quality KPIs, ...) can be also returned in form of timeseries results, with a default time aggregation step of one hour. The word *timeseries* inside the methods names allows to identify these KPIs as a particular category that will generate a different output named *data_res_timeseries.csv* containing timestamps as indexes and KPIs results in different columns. Considering the nature of this file, it is generated separately for each simulation in a devoted subfolder (together with graphs). Timeseries results can be useful to both deepen the behaviour of specific simulated building conditions and to give a visual idea of the simulated variables trend through interfaces.

timeseries_q_c(df, timestep='1H', gen_factor=1, ep_factor=1)

Compute timeseries values (sum over the time period) of primary energy need for cooling in kWh/m2, applying correction factors for generation losses, then multiplying by primary energy factor. Corrections for emission, regulation and distribution losses are considered previously applied.

Parameters

- **df** (class:*pandas.core.frame.DataFrame*) – Dataframe which must contain at “Q_c[kWh/m2]” and “Date/Time” columns.
- **timestep** (*str*, *default* = “1H”) – time aggregation required
- **gen_factor** (*float*, *default* 1) – correction factor for generation losses, in range [0,1].
- **ep_factor** (*float*, *default* 1) – primary energy conversion factor, in range [0,1].

Returns

DataFrame of cooling primary energy values over time

Return type

class:*pandas.core.frame.DataFrame*

timeseries_co2(df, timestep='1H')

Compute timeseries of indoor CO2 concentration (ppm) according to requested time aggregation (average on the time period).

Parameters

- **timestep** (*str*, *default* = “1H”) – time aggregation required
- **df** (class:*pandas.core.frame.DataFrame*) – Dataframe which must contain at least CO2 concentration indoor “CO2_i[ppm]” and “Date/Time” columns.

Returns

DataFrame of indoor co2 values over time

Return type

class:*pandas.core.frame.DataFrame*

```
timeseries_acm_hourly_cat(df, eu_norm='16798-1:2019', alpha=0.8, filter_by_occupancy=0, when={})
```

Compute hourly ACM category according to different European normatives. Filters according to occupancy or dates can be applied. ACM category is expressed in terms of distance from comfort central line.

Parameters

- **df** (class:*pandas.core.frame.DataFrame*) – dataframe should contain “Date/Time” column in format ‘year/month/day hour:minutes:seconds’, “T_db_o[C]” preferably with a subhourly timestep and “T_op_i[C]”. Optional “Occupancy” column accepting only 0/1 values.
- **eu_norm** (*str, optional*) – It can be set to ‘15251:2007’ if old UE norm computation is desired, defaults to ‘16798-1:2019’.
- **alpha** (*float, optional*) – With old UE norm ‘15251:2007’ alpha is a free parameter in range [0,1), defaults to 0.8
- **filter_by_occupancy** (*int, optional*) – It can be set 0 or 1, depending on whether activate occupancy filtering on thermal comfort KPIs computation or not, default 0.
- **when** (*dict, optional*) – dictionary with ‘start’ and ‘end’ keys and values in format ‘year/month/day hour:minutes:seconds’

Returns

Dataframe containing “Date/Time” and “dist” columns.

Return type

class:*pandas.core.frame.DataFrame*

Concerning graphs only devoted KPIs, carpet plots are the main example of strictly visual indicators and analysis. The same function is able to receive in input different environmental variables and adapt graph generation to the different cases, e.g., indoor and outdoor dry bulb temperature, CO2, distance from adaptive comfort model category I central line. Examples are shown in Figure 18.

```
carpet_plot(df, variable, title=None)
```

Generate a carpet plot

Parameters

- **df** (class:*pandas.core.frame.DataFrame*) – DataFrame which must contain Date/Time and variable column.
- **variable** (*str*) – variable name according to standardized nomenclature
- **title** (*str, optional*) – Title of the figure, defaults to None

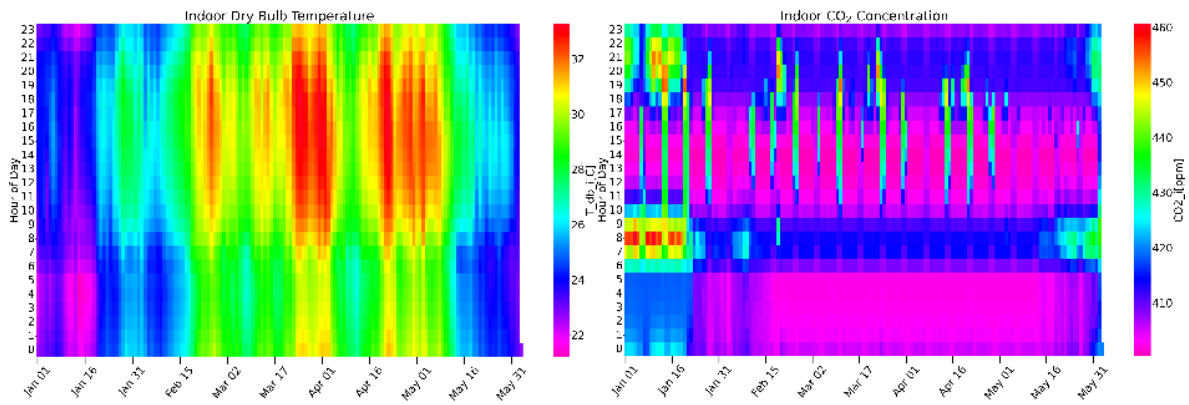


Figure 18 Example of carpet plots

4.4 Scenarios

In the following paragraphs, PREDYCE scenarios are briefly listed and described, with particular attention to sensitivity analysis scenario. Then, in section 4.5 examples of usage are reported. As previously detailed in section 4.1 and 4.2, PREDYCE actions and functionalities are exploited combining them in different scripts, such allowing to perform different tasks. At present, the different scripts are treated as separate applications, each potentially executable both locally and remotely, but in the future they can be unified under the same application and recognized through a scenario name keyword inside the input JSON file. Different PREDYCE scenarios can be exploited to act both on simulated and/or monitored data.

4.4.1 Sensitivity analysis

Sensitivity analysis is the base and most powerful PREDYCE scenario, since any other is somehow based on it. It allows performing parametric analysis by automatically modifying the base building model according to permutations of all parameters listed in the input JSON file and computing requested KPIs. Other scenarios may exploit different logics in the choice or combination of parameters, but at the base there is PREDYCE capability of executing parallel runs of EnergyPlus simulations, storing results in aggregated and lightweight form. At present, analysis on best values combination (e.g., to find minimum energy consumption) has to be done next by looking (also with scripting) at all results contained in *data_res.csv*, while it is not implemented yet an optimization procedure through optimization algorithms or surrogate modelling in order to accelerate space of parameters scanning and find an optimal building set up. Anyhow, the future integration of optimization procedures, also exploiting the many already existing powerful Python libraries devoted to this task, is made possible by the tool modularity.

Since sensitivity analysis is the base PREDYCE scenario, it can give a clear idea on how a scenario can be run both locally and remotely. Figure 19 shows all available command line options: besides mandatory inputs, many other arguments can be set or specified, for example desired directories for outputs (-d) and plots (-p), the path of the IDD file, whether to include or not the base building model with no modifications in the permutation of parameters (-o), the number of CPUs to be used (-j) which are otherwise automatically set to the maximum allowed by the used machine. Among the arguments it can be also specified the path of monitored data file (-m), but this is unused for sensitivity analysis.

```
usage: run_sensitivity_analysis.py [-h] [--checkpoint-data CHECKPOINT DATA]
                                [-c CHECKPOINT INTERVAL] [-d OUTPUT DIRECTORY]
                                [-i IDD VERSION] [-f INPUT FILE] [-j JOBS] [-o]
                                [-m MEASURED DATA] [-p PLOT DIRECTORY]
                                [-t TEMP DIRECTORY] [-w WEATHER] model
```

Figure 19 Sensitivity analysis command line options

Figure 20 instead shows how the same script can be launched remotely through a basic web interface. In this case just mandatory inputs can be inserted while all other settings are handled automatically on the web service running on the server. Also, server-to-server communication can be exploited by sending REST requests by a code without the need of the graphical interface. The latter approach is the one adopted during the WP4 implementation actions.

Figure 20 Sample web interface for remote run

As previously said, at present sensitivity analysis requires a not automated post-analysis step both to find optimal building conditions and to deepen meaning of obtained results. Despite in the future automatic procedures may be implemented, the freedom of analysis guaranteed by this scenario allows methodology definition also for other more specific use cases. For example, Figure 21 shows simple graphs obtained as a post-analysis on aggregated results contained in the same *data_res.csv*: these kinds of graphs can be used as a base to further create a new *retrofit* scenario, suggesting solutions according to different combinations of criteria (e.g., integrating also cost analysis).

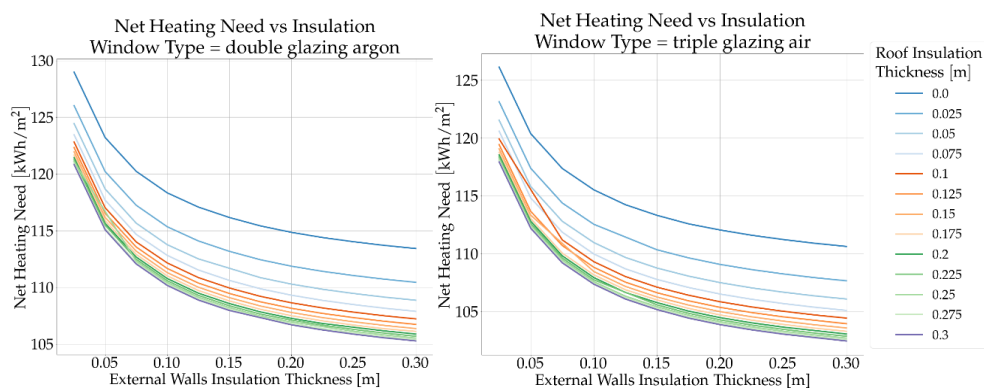


Figure 21 Example of post-analysis for sensitivity analysis application to retrofit use case applied to one of the TPM residential building

4.4.2 Other scenarios

Two other scenarios have been developed specifically for E-DYCE purposes and are here just briefly listed, while a detailed description can be found in correlated deliverable D3.2. Particularly, they differ from sensitivity analysis for the ability of integrating monitored data in the loop.

- **Model verification:** it is a semi-automatic scenario since it requires at present a human interpretation in order to choose how to launch it (e.g., which parameters to include in the calibration process and in which range) and how many times (e.g., the step). It allows to choose among a pre-defined set of building parameters minimizing RMSE and MBE on defined variables (e.g., consumption or internal temperatures), comparing simulation and monitored data from the field. It exploits PREDYCE ability to compute KPIs also on monitored data and to generate EPW from real weather data, which is included as an extra module in the Python tool.
- **Performance gap:** it is one of the most significative scenarios for E-DYCE project since it allows to return the performance gap between calibrated simulations performed under standard and/or more realistic building settings versus monitored actual building behaviour.

5 Application testing

A basic sensitivity analysis application case is proposed in this chapter to highlight PREDYCE functionalities and organisation. The example can be launched from remote through the devoted REST API or a dedicated [web interface](#)², by asking to POLITO research unit temporary credentials available for this purpose³. Also input files to be used can be found in a [shared folder](#)⁴ together with outputs already generated. The generation of the outputs may take several minutes depending on many factors; the output files are then retrieved in the Download folder of the browser whenever the web interface is used. In the following the example will be explained in detail in order to make easier the understanding of folder content.

For an example purpose it was used as building model a typical residential flat illustrated in Figure 22, part of a multi-storey building, in line with residential building typologies suggested in architectural manuals - see for example (Neufert et al., 2013). The building is structured with two flats in each floor, while just a single unit is simulated. The simulated unit is considered to be at an intermediate level, with upper, lower and adjacent units working at the same temperature. Upper-floor balconies are also included in order to consider shading effects. The IDF file of this building model can be found in the provided folder.

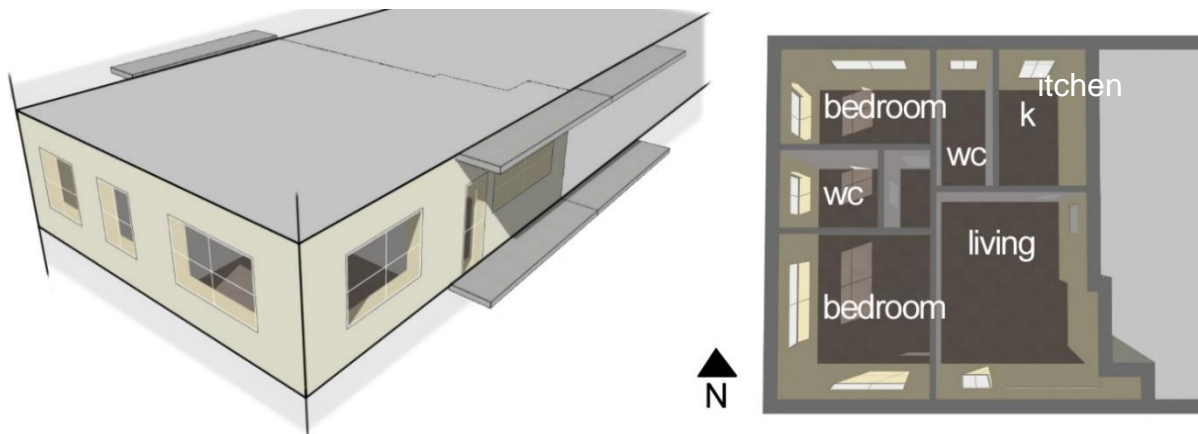


Figure 22 Sample residential unit

Concerning weather file, the Meteonorm-produced TMY of Casaccia (Rome), where ENEA living LAB is located, is considered in this example and can be found in the folder. The input JSON file instead is structured to test the impact of different retrofit solutions on energy consumption over the whole year. In Figure 23 the part of input file including the tested actions is reported. An insulation layer is added both externally to boundary walls and to the ceiling with three thicknesses (expressed in meters): the material names correspond with objects stored in internal database of IDF objects. Moreover, both windows to wall ratio and windows type are varied.

² <http://130.192.20.228:3200/sa>

³ You can send an e-mail to giacomo.chiesa@polito.it, paolo.grasso@polito.it, francesca.fasano@polito.it

⁴ <https://www.dropbox.com/sh/dwpexstrky7lqlb/AABpMwIYNF-mW95PqAXVEDeda?dl=0>

```
{
  "actions": {
    "add_external_insulation_walls": {
      "ins_data": [[XPS Extruded Polystyrene CO2 Blowing", "Gypsum Plastering"]],
      "Thickness": [0.05, 0.15, 0.25]
    },
    "add_insulation_flat_roof": {
      "ins_name": ["MW Glass Wool (rolls)"],
      "Thickness": [0.05, 0.15, 0.25]
    },
    "change_windows_system": {
      "win_type": ["double glazing argon", "triple glazing air"]
    },
    "change_wwr": {
      "wwr_value": [0.2, 0.4, 0.6]
    }
  }
}
```

Figure 23 Input JSON actions

The computed KPIs are reported in Figure 24 and include final and primary energy needs for heating and cooling, primary energy needs for lighting, energy signature of heating and cooling net energy needs and finally carpet plots of PPD (Predicted Percentage of Dissatisfied) for each parametric simulation. Concerning consumptions, loss factors have been chosen according to typical Italian buildings conditions, in line with tables in the UNI-TS 11300 series and UNI-EN 15316, and assuming non-renewable electric sources for cooling and lighting, while natural gas for heating - primary non-renewable energy factors are retrieved by the UNI EN 15603 annex E. Also, timeseries results with default hourly timestep are requested as KPIs for heating and cooling primary energy needs.

```
"kpi": {
  "Q_c": {"em_factor": 0.95, "reg_factor": 0.95, "distr_factor": 1,
    "gen_factor": 2.5, "ep_factor": 3.14},
  "Q_h": {"em_factor": 0.97, "reg_factor": 0.94, "distr_factor": 0.97,
    "gen_factor": 1, "ep_factor": 1.36},
  "Q_I": {"ep_factor": 3.14},
  "f_Q_c": {"em_factor": 0.95, "reg_factor": 0.95, "distr_factor": 1,
    "gen_factor": 2.5},
  "f_Q_h": {"em_factor": 0.97, "reg_factor": 0.94, "distr_factor": 0.97,
    "gen_factor": 1},
  "energy_signature": {},
  "timeseries_Q_c": {},
  "timeseries_Q_h": {},
  "carpet_plot": {"variable": "ppd"}
}
```

Figure 24 Input JSON KPIs

In main output folder *data_res.csv* file can be found. Parts of it are reported in Figure 25 and Figure 26, focusing on input and output columns separately. Each row in the file corresponds to a specific combination of input parameters (insulation thickness, window type, etc...) and it is identified by a unique index number. Numbers are not ordered since all simulations were executed in parallel, so order reflects simulations ending time. Besides input descriptive columns, calculated KPIs are attached: they can have numeric or dictionary form depending on the quantity of information they carry. Simple post-analysis can be easily performed directly acting on the CSV file (e.g., in Microsoft Excel), for example ordering results by lower total consumption values. Together with aggregated results, for each simulation a folder named

with row index identifier is also generated and contains both timeseries results and plots. Figure 27 shows an example of *data_res_timeseries.csv*: hourly consumption values are associated to correspondent timestamp for the simulation period.

	add_external_insulation_walls.Thickness	add_insulation_flat_roof.Thickness	change_windows_system.win_type	change_wwr.wwr_value
0	0.05	0.05	double glazing argon	0.2
1	0.05	0.05	double glazing argon	0.4
2	0.05	0.05	double glazing argon	0.6

Figure 25 Example of data_res.csv inputs part

	Q_c	Q_h	Q_l	f_Q_c	f_Q_h	energy_signature
0	83.20	68.50	40.96	26.49	50.36	{"1D": {"cooling": {"deltaT": [-2.43, 10.13], "cooling": [9.12, -1.99]}, "heating": {"deltaT": [-2.43, 16.41], "heating": [-0.46, 1.36]}}, "2D": {"cooling": {"deltaT": [-2.43, 10.13, -2.43, 10.13], "solarRadiation": [277.09, 277.09, 514.22, 514.22], "cooling": [7.55, -2.55, 9.54, -0.56]}, "heating": {"deltaT": [-2.43, 16.41, -2.43, 16.41], "solarRadiation": [99.12, 99.12, 514.22, 514.22], "heating": [0.57, 1.51, -0.55, 0.38]}}
1	161.1376326	66.25480436	39.98722088	51.31771739	48.71676791	< ... >
2	241.3111602	67.5757633	39.64473542	76.85068797	49.68806125	< ... >

Figure 26 Example of data_res.csv outputs part

Date/Time	timeseries_Q_c	timeseries_Q_h
01/01 00:00	0	0.00840
01/01 01:00	0	0.00988
01/01 02:00	0	0.00843
01/01 03:00	0	0.00877
01/01 04:00	0	0.00907
01/01 05:00	0	0.00937

Figure 27 Example of data_res_timeseries.csv

Figure 28 and Figure 29 report energy signatures for best and worse parametric choice in terms of total primary energy need. Best case results to be the most insulated one with lower WWR, while worst case the less insulated with higher WWR. Figure 30 instead shows PPD carpet plots for both cases, showing a reduction of dissatisfied particularly in mid-seasons. Other single simulation-based graphs can be also generated through PREDYCE in order to deepen post-analysis of aggregated results.

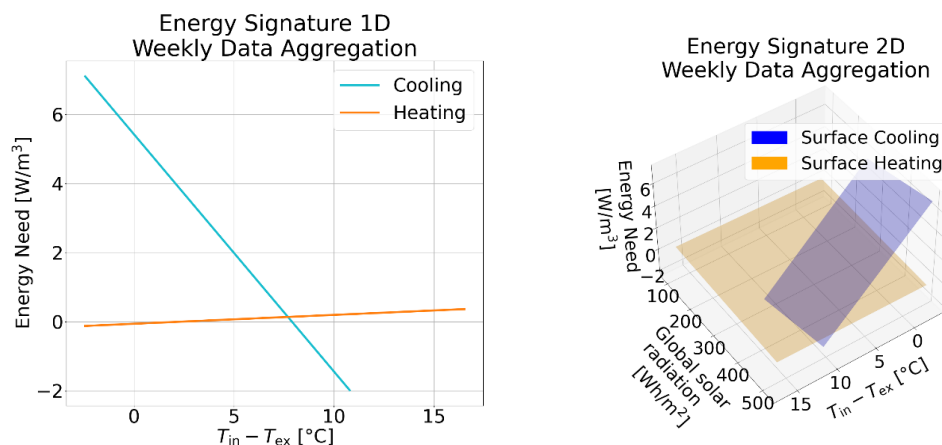


Figure 28 Energy signature best case (folder 51)

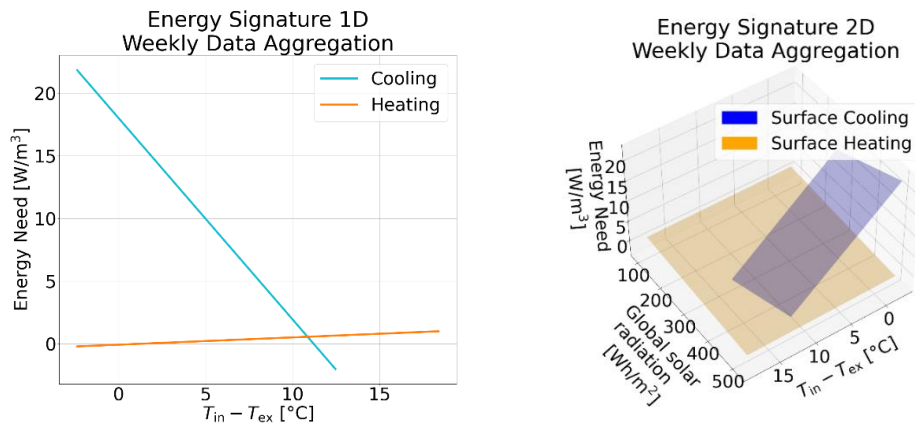


Figure 29 Energy signature worst case (folder 2)

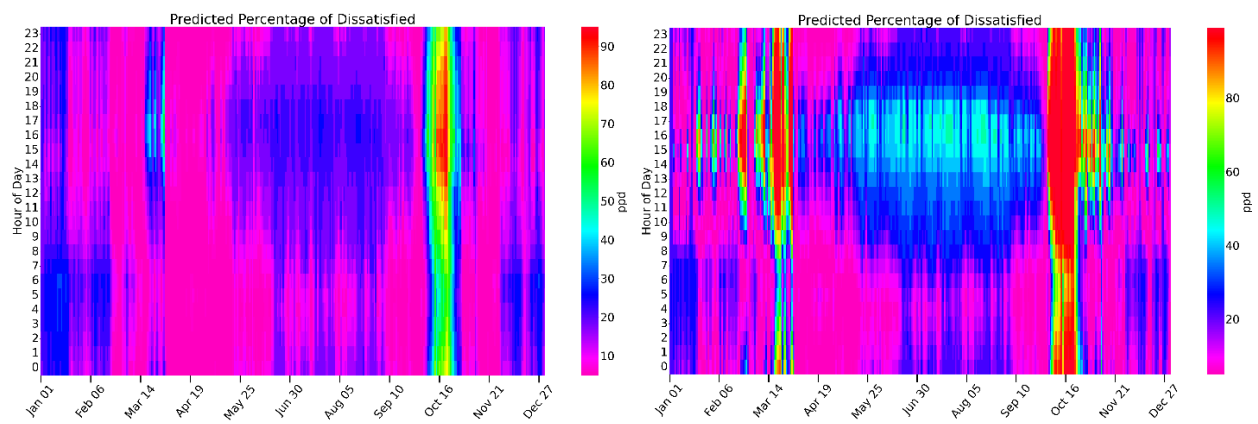


Figure 30 PPD carpet plots for best (a) and worse (b) cases

6 Conclusions and Outlook

D3.1 includes, such as mentioned in the introductive section, this report detailing methodological and specific descriptions correlated to the development of a sample dynamic simulation platform supporting E-DYCE functionalities and communicating with the E-DYCE middleware (based on Fusix). Advanced functionalities connected to the integration of the dynamic simulation platform in the E-DYCE middleware are discussed in related deliverables from WP3 and WP4. Current PREDYCE version adopted for E-DYCE purposes bases on EnergyPlus v8.9 and compatibility with v9.x is not assured due to changes in the correlated EnergyPlus IDD files. Additionally, the development of actions and KPIs is expected to continue during next project phases thanks to outcomes of other WPs and in particular WP2 and WP4 and thanks to the applications of the above-mentioned solutions to project demo buildings (WP5).

Nevertheless, the test of result compatibility between EnergyPlus and DIAL+ shows that it is possible to be “software neutral” considering the potential overlapping between initial inputs and retrieved outputs, opening the possibility, during a commercial deployment phase, to apply the E-DYCE methodology to other dynamic tools by including different dynamic simulation platforms.

Initial results from ‘DYCE’ development action (D3.1 and D3.2) are reported in (Chiesa et al., 2021), while additional dissemination actions are expected during next months.

7 Bibliography

DesignBuilder Software, 2020. DesignBuilder. DesignBuilder Software.

DOE, NREL, 2020. EnergyPlus™. DOE, BTO, NREL.

EDYCE, 2021. D1.2 Operational dynamic Energy Performance Certificate (EPC) specifications (Project report No. D1.2). PoliTO.

Neufert, E., Di Giuda, G.M., Villa, V., Gottfried, A., Piantanida, P., 2013. Enciclopedia pratica per progettare e costruire: manuale a uso di progettisti, costruttori, docenti e studenti : fondamenti, norme e prescrizioni per progettare, costruire, dimensionare e distribuire a misura d'uomo. U. Hoepli, Milano.

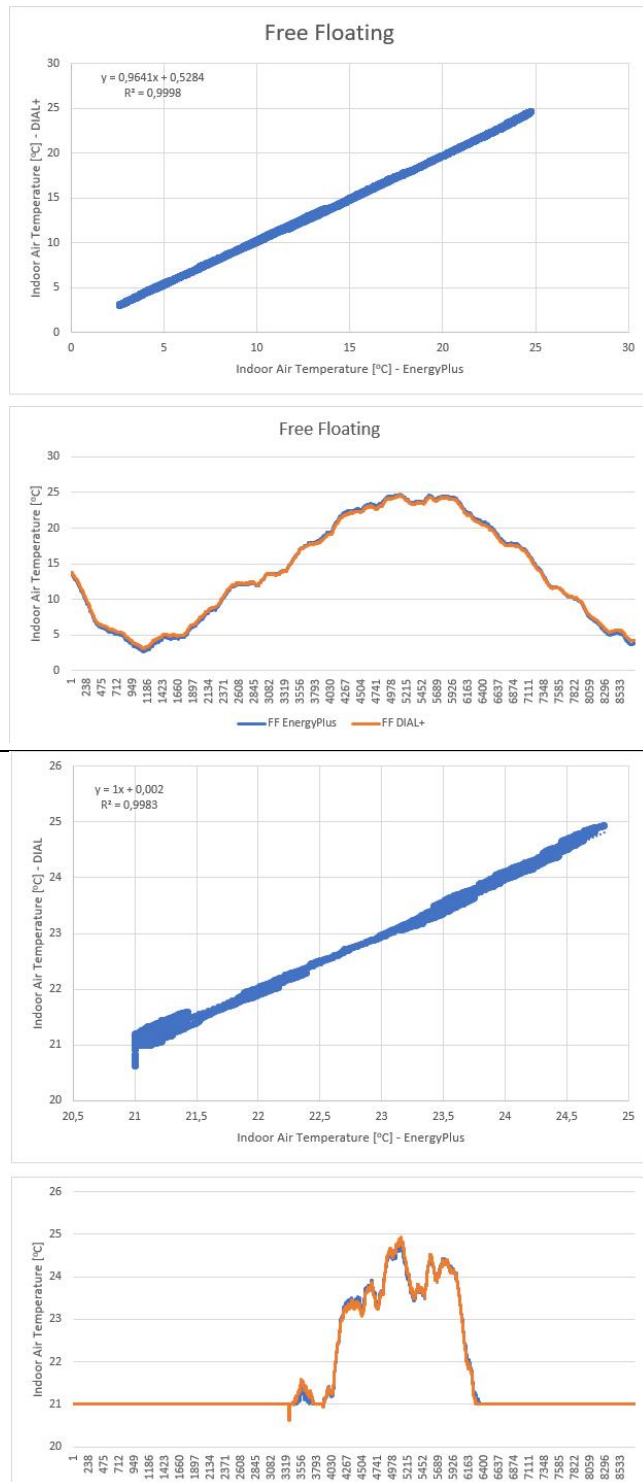
NREL, ANL, LBNL, ORNL, PNNL, 2021. OpenStudio.

Roudsari, M.S., 2013. Honeybee. Ladybug Tools LLC.

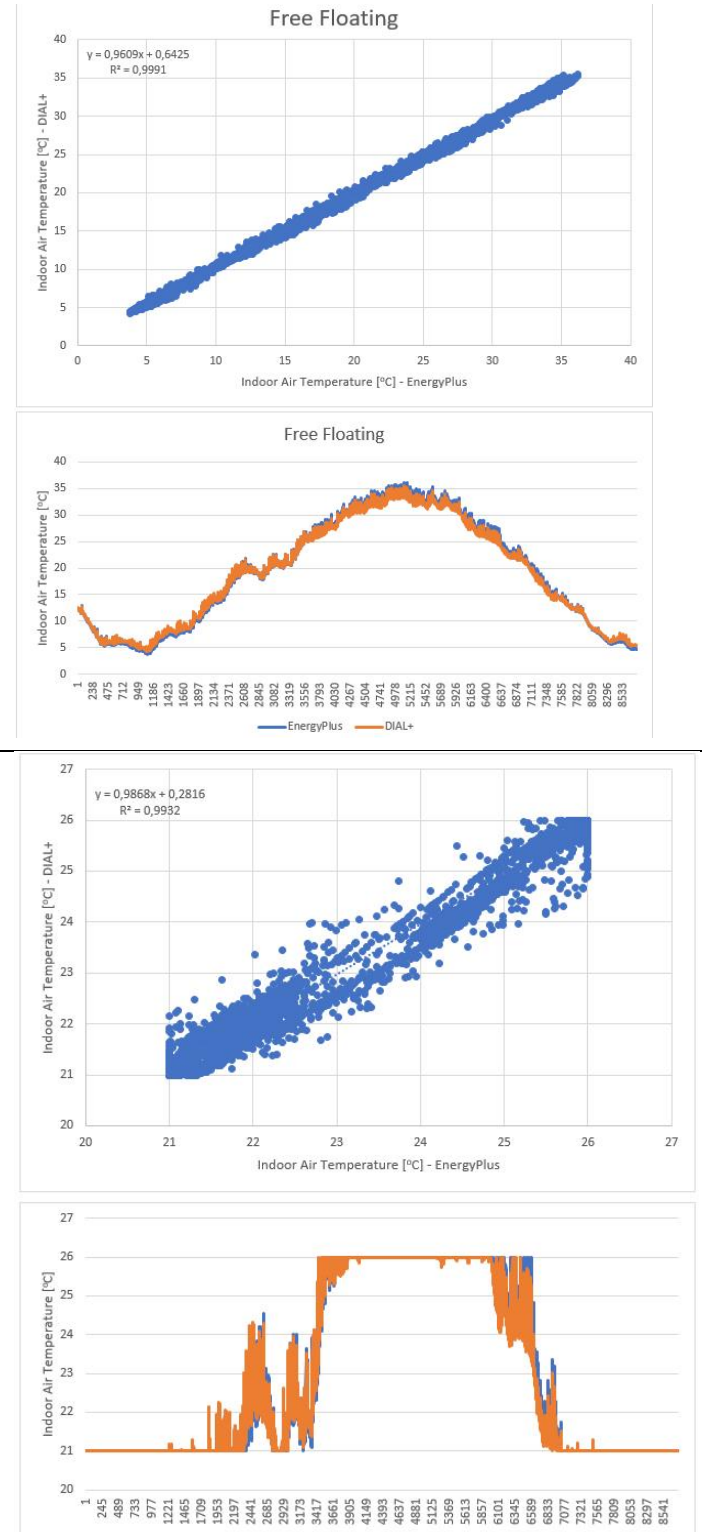
8 Annex A⁵

Heavyweight – High thermal mass

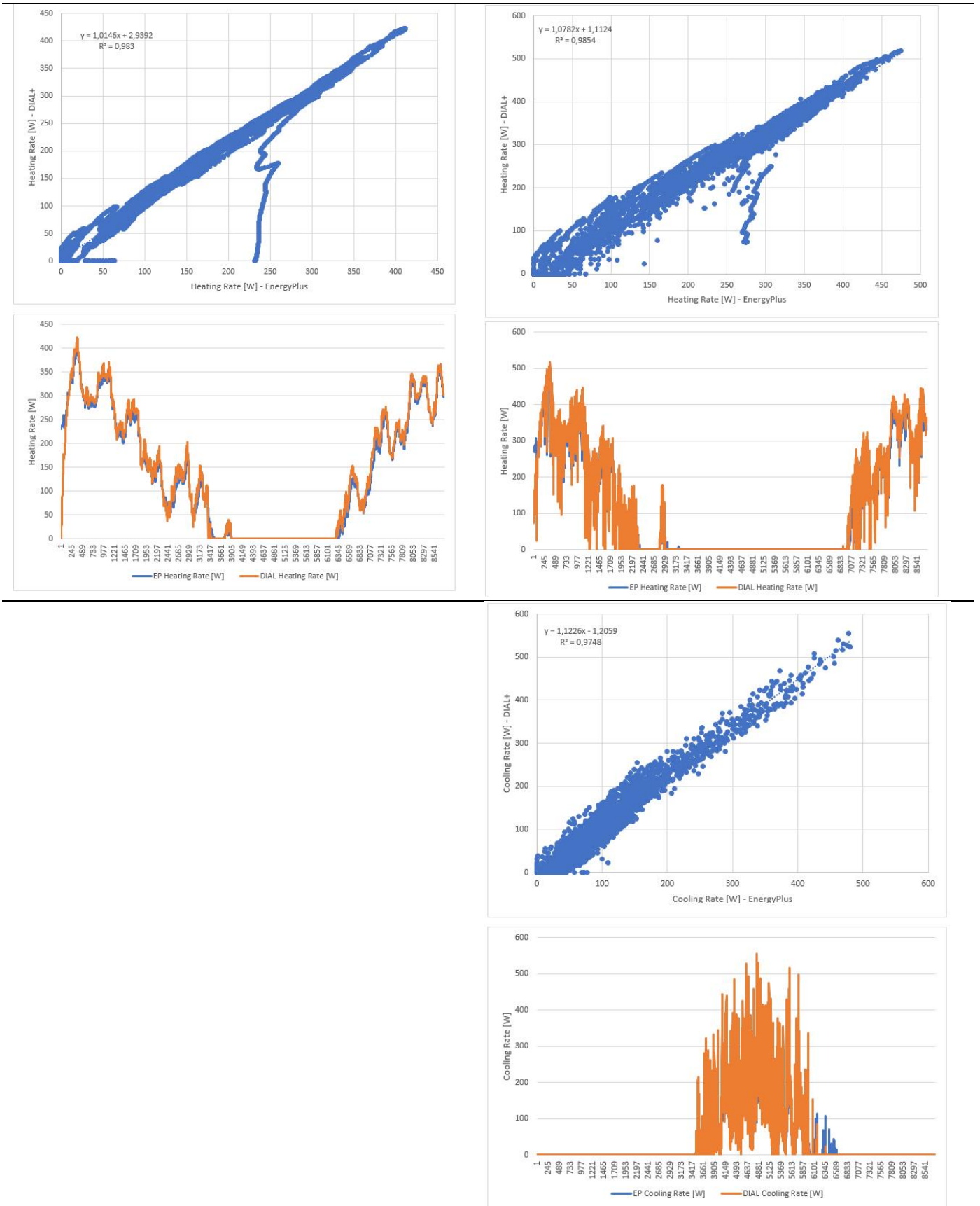
Without window



With window

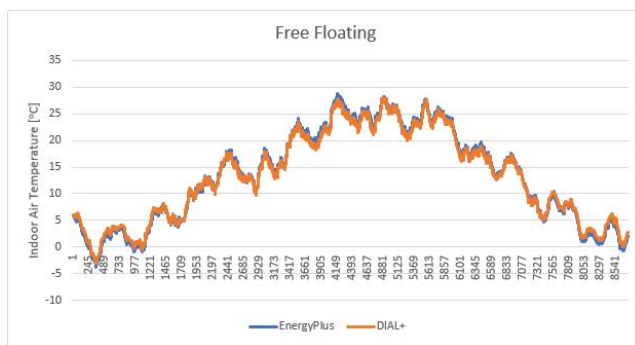
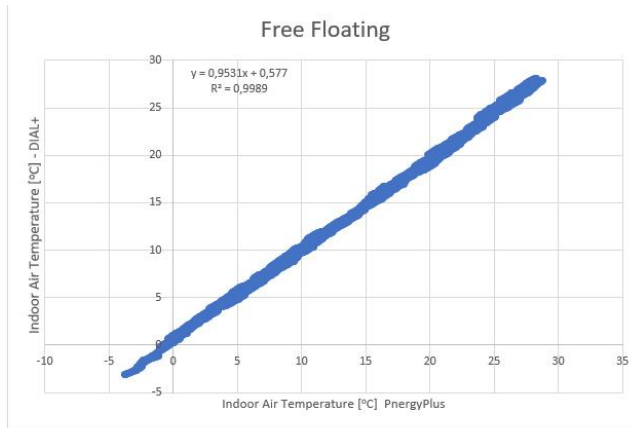


⁵ This Annex is developed by ESTIA



Lightweight – Low thermal mass

Without window



With window

