Dibs Documentation

Version

Table of Contents

Contents:

Welcome to Dibs Documentation's documentation!

src

dibs_computing_core package

Subpackages

dibs_computing_core.iso_simulator package

Subpackages

dibs_computing_core.iso_simulator.building_simulator package

Submodules

dibs_computing_core.iso_simulator.building_simulator.simulator module

this class implements the business logic to simulate a given building

class dibs_computing_core.iso_simulator.building_simulator.simulator. BuildingSimulator (*datasource : DataSource*)

Bases: `object`

absorption_refrigeration_system ( ) → bool

Checks if cooling supply system of the building is AbsorptionRefrigerationSystem

Returns:
True or False

Return typ
boolean

air_cool ( ) → bool

Checks if cooling supply system of the building in the list named air_cool Returns:

True or False

Return typ
boolean

biogas_boiler_types ( ) → bool
Checks if heating supply system of the building in the list named biogas_boiler_types Returns:

True or False

Return typ
boolean

biogas_oil_boilers_types ( ) → bool
Checks if heating supply system of the building in the list named biogas_oil_boilers_types Returns:

True or False

Return typ
boolean

build_east_window ( ) → Window
This method builds a window object Returns:

window

Return type
Window

build_north_window ( ) → Window
This method builds a window object Returns:

window

Return type
Window

build_south_window ( ) → Window
This method builds a window object Returns:

window

Return type
Window

build_west_window ( ) → Window

This method builds a window object

Returns:

window

Return type

Window

build_windows_objects ( ) → List [Window]

This method builds a list of all windows (south, west, east and north) Returns:

windows

Return type

List[Window]

calc_altitude_and_azimuth (*hour : int*) → Tuple [float , float]

Call calc_sun_position(). Depending on latitude, longitude, year and hour - Independent from epw weather_data Args:

hour: hour to simulate

Returns:

altitude, azimuth

Return type

Tuple[float, float]

calc_appliance_gains_demand (*occupancy_schedule : List [ScheduleName]* , *appliance_gains : float* , *hour : int*) → float

Calculate appliance_gains as part of the internal_gains Args:

occupancy_schedule: schedule name appliance_gains: hour: hour to simulate

Returns:

appliance_gains_demand

Return type

float

calc_building_h_ve_adj (*hour : int* , *t_out : float* , *usage_start : int* , *usage_end : int*) → float

Calculate H_ve_adj, See building_physics for details Args:

hour: hour to simulate t_out: Outdoor air temperature [C] usage_start: Beginning of usage time according to SIA2024 usage_end: Ending of usage time according to SIA2024

Returns:
h_ve_adj

Return type
float

calc_energy_demand_for_time_step (*internal_gains : float* , *t_out : float* , *t_m_prev : float*) → None

Calculate energy demand for the time step Args:

internal_gains: internal heat gains from people and appliances [W] t_out: Outdoor temperature of this timestep t_m_prev: Previous air temperature [C]

Return type:
None

calc_gains_from_occupancy_and_appliances (*occupancy_schedule : List [ScheduleName]* , *occupancy : float* , *gain_per_person : float* , *appliance_gains : float* , *hour : int*) → float

Calculate gains from occupancy and appliances This is thermal gains. Negative appliance_gains are heat sinks! Args:

occupancy_schedule: schedule name occupancy: Occupancy [people] gain_per_person: appliance_gains: hour: hour to simulate

Returns:
internal_gains

Return type
float

calc_hot_water_usage (*occupancy_schedule : List [ScheduleName]* , *tek_dhw_per_occupancy_full_usage_hour : float* , *hour : int*) → Tuple [float , float , float , float]

Calculate hot water usage of the building for the time step with (self.datasource.building.heating_energy / self.datasource.building.heating_demand) represents the Efficiency of the heat generation in the building Args:

occupancy_schedule: tek_dhw_per_occupancy_full_usage_hour: hour:

Returns:
hot_water_demand, hot_water_energy, hot_water_sys_electricity, hot_water_sys_fossils

Return type
Tuple[float, float, float, float]

calc_illuminance_for_all_windows (*sun_altitude : float* , *sun_azimuth : float* , *hour : int*) → None
Calculates the illuminance in the building zone through the set window Args:

sun_altitude: Altitude Angle of the Sun in Degrees sun_azimuth: Azimuth angle of the sun in degrees hour: hour to simulate

Return type:
None

calc_occupancy (*occupancy_schedule : List [ScheduleName]* , *hour : int*) → float
Calc occupancy for the time step Args:

occupancy_schedule: schedule name hour: hour to simulate

Returns:
occupancy

Return type
float

calc_solar_gains_for_all_windows (*sun_altitude : float* , *sun_azimuth : float* , *t_air : float* , *hour : int*) → None
Calculates the solar gains in the building zone through the set window Args:

sun_altitude: Altitude Angle of the Sun in Degrees sun_azimuth: Azimuth angle of the sun in degrees t_air: hour: hour to simulate

Return type:
None

calc_sum_illuminance_all_windows ( ) → float
Sum of transmitted illuminance of all windows Returns:

transmitted illuminance_sum

Return type
float

calc_sum_solar_gains_all_windows ( ) → float
Sum of solar gains of all windows Returns:

solar_gains_sum

Return type

float

check_cooling_system_elctricity_sum (*calculation_of_sum : CalculationOfSum* , *f_hs_hi : float* , *f_ghg : int* , *f_pe : float*) → Tuple [int , float , float , float]

Args:

calculation_of_sum: sum object f_hs_hi: f_ghg: f_pe:

Returns:

cooling_sys_electricity_hi_sum, cooling_sys_carbon_sum, cooling_sys_pe_sum, cooling_sys_fossils_hi_sum

Return type

Tuple[int, float, float, float]

check_energy_area_and_heating ( )

If there's no heated area (energy_ref_area == -8) or no heating supply system (heating_supply_system == 'NoHeating') no heating demand can be calculated. In this case skip calculation and proceed with next building. Returns:

check_heating_sys_electricity_sum (*calculation_of_sum : CalculationOfSum* , *f_hs_hi : float* , *f_ghg : int* , *f_pe : float*) → Tuple [int , float , float , float]

Args:

calculation_of_sum: sum object f_hs_hi: f_ghg: f_pe:

Returns:

heating_sys_electricity_hi_sum, heating_sys_carbon_sum, heating_sys_pe_sum, heating_sys_fossils_hi_sum

Return type

Tuple[int, float, float, float]

check_hotwater_sys_electricity_sum (*calculation_of_sum : CalculationOfSum* , *f_hs_hi : float* , *f_ghg : int* , *f_pe : float*) → Tuple [int , float , float , float]

Args:

calculation_of_sum: sum object f_hs_hi: f_ghg: f_pe:

Returns:

hot_water_sys_electricity_hi_sum, hot_water_sys_pe_sum, hot_water_sys_carbon_sum, hot_water_sys_fossils_hi_sum

Return type

Tuple[int, float, float, float]

check_if_central_dhw_use_same_fuel_type_as_heating_system (*fuel_type*) → str

Checks if central dhw uses the same fuel type as the heating system Assumption: Central DHW-Systems use the same Fuel_type as Heating-Systems, only decentral DHW-Systems might have another Fuel-Type Args:

fuel_type: fuel type

Returns:
fuel_type

Return type
str

check_if_central_heating_or_central_dhw ( ) → bool
Checks if dhw system of the building in the list named central Returns:

True or False

Return typ
boolean

check_if_heat_pump_air_or_ground_source ( ) → bool
Checks if dhw system of the building in the list named heat_source Returns:

True or False

Return typ
boolean

choose_cooling_energy_fuel_type ( ) → str | GHGEmissionError
Returns:
fuel_type or throws an error

Return type
Union[str, GHGEmissionError]

choose_the_fuel_type ( ) → str
Choose the fuel type based on the heating_supply_system of the building Returns:

fuel_type

Return type
str

coal_solid_fuel_boiler ( ) → bool
Checks if heating supply system of the building is CoalSolidFuelBoiler Returns:

True or False

Return typ
boolean

cooling_sys_hi_sum (*cooling_sys_electricity_hi_sum : int* , *cooling_sys_fossils_hi_sum : float*) → float

Calculates sum of cooling_sys_electricity_hi_sum and cooling_sys_fossils_hi_sum Args:

cooling_sys_electricity_hi_sum: cooling_sys_fossils_hi_sum:

Returns:
cooling_sys_electricity_hi_sum + cooling_sys_fossils_hi_sum

Return type
float

direct_heater ( ) → bool
Checks if heating supply system of the building is DirectHeater Returns:

True or False

Return typ
boolean

district_cooling ( ) → bool
Checks if cooling supply system of the building is DistrictCooling Returns:

True or False

Return typ
boolean

district_heating ( ) → bool
Checks if heating supply system of the building is DistrictHeating Returns:

True or False

Return typ
boolean

electric_heating ( ) → bool
Checks if heating supply system of the building is ElectricHeating Returns:

True or False

Return typ
boolean

extract_outdoor_temperature (*hour : int*) → float
Extract the outdoor temperature in building_location for that hour from weather_data Args:

hour: hour to simulate

Returns:

outdoor_temperature

Return type

float

extract_year (*hour : int*) → int

Extract the year based on a given hour Args:

hour: hour to simulate

Returns:

year

Return type

int

first_natural_gas ( ) → bool

Checks if heating supply system of the building in the list named lgaz Returns:

True or False

Return typ

boolean

gas_boiler_standard ( ) → bool

Checks if heating supply system of the building in the list named gas_boiler_standard Returns:

True or False

Return typ

boolean

gas_chip ( ) → bool

Checks if heating supply system of the building is GasCHP Returns:

True or False

Return typ

boolean

gas_engine_piston_scroll ( ) → bool

Checks if cooling supply system of the building is GasEnginePistonScroll Returns:

True or False

Return typ

boolean

get_appliance_gains_elt_demand (*occupancy_schedule : List [ScheduleName]* , *appliance_gains : float* , *hour : int*) → float

Appliance_gains equal the electric energy that appliances use, except for negative appliance_gains of refrigerated counters in trade buildings for food! The assumption is: negative appliance_gains come from referigerated counters with heat pumps for which we assume a COP = 2. Args:

occupancy_schedule: schedule name appliance_gains: hour: hour to simulate

Returns:
appliance_gains_elt_demand

Return type
float

get_conversion_factor_heating (*fuel_type : str*)

Umrechnungsfaktor von Brennwert (Hs) zu Heizwert (Hi) einlesen Args:

fuel_type: fuel_type

Returns:
relation_calorific_to_heating_value_GEG

get_ghg_factor_heating (*fuel_type : str*)

GHG-Factor Heating Args:

fuel_type: fuel_type

Returns:
gwp_specific_to_heating_value_GEG

get_ghg_pe_conversion_factors (*fuel_type*)

get_pe_factor_heating (*fuel_type : str*)

PE-Factor Heating Args:

fuel_type: fuel_type

Returns:
primary_energy_factor_GEG

get_schedule ( ) → Tuple [List [ScheduleName] , str , float] | ValueError

Find occupancy schedule from SIA2024, depending on hk_geb, uk_geb from csv file Returns:

list_of_schedule_name, schedule_name or throws an error

Return type

Union[Tuple[List[ScheduleName], str, float], ValueError]

get_tek ( ) → Tuple [float , str] | ValueError

Find TEK values from Partial energy parameters to build the comparative values in accordance with the announcement of 15.04.2021 on the Building Energy Act (GEG) of 2020, depending on hk_geb, uk_geb Returns:

tek_dhw, tek_name or throws an error

Return type

Union[Tuple[float, str], ValueError]

get_usage_start_and_end ( ) → Tuple [int , int]

Find building's usage time DIN 18599-10 or SIA2024 Returns:

usage_start, usage_end

Return type

Tuple[int, int]

get_weather_data ( ) → List [WeatherData]

This method retrieves the right weather data according to the given weather_period and file_name

Returns:

weather_data_objects

Return type

List[WeatherData]

hard_coal ( ) → bool

See above method coal_solid_fuel_boiler() and solid_fuel_liquid_fuel_furnace() for more information Returns:

True or False

Return type

bool

heat_pump ( ) → bool

Checks if heating supply system of the building in the list named heat_pumping Returns:

True or False

Return typ

boolean

heat_pump_or_electric_heating ( ) → bool

See above method heat_pump() and electric_heating() for more information Returns:

True or False

Return type

bool

hot_energy_hi_sum (*hotWater_sys_electricity_hi_sum : int* , *hot_water_sys_fossils_hi_sum : float*) → float

Calculates sum of hotWater_sys_electricity_hi_sum and hot_water_sys_fossils_hi_sum Args:

hotWater_sys_electricity_hi_sum: hot_water_sys_fossils_hi_sum:

Returns:

hotWater_sys_electricity_hi_sum + hot_water_sys_fossils_hi_sum

Return type

float

lgas_boiler_temp ( ) → bool

Checks if heating supply system of the building in the list named lgas_boiler_temp Returns:

True or False

Return typ

boolean

no_cooling ( ) → bool

Checks if cooling supply system of the building is NoCooling Returns:

True or False

Return typ

boolean

no_heating ( ) → bool

Checks if heating supply system of the building is NoHeating Returns:

True or False

Return typ

boolean

oil_boiler_types ( ) → bool

Checks if heating supply system of the building in the list named oil_boiler_types Returns:

True or False

Return typ
boolean

set_t_air_based_on_hour (*hour : int*) → float

Define t_air for calc_solar_gains(). Starting condition (hour==0) necessary for first time step

Args:
hour: hour to simulate

Returns:
t_air

Return type
float

solid_fuel_liquid_fuel_furnace ( ) → bool

Checks if heating supply system of the building is SolidFuelLiquidFuelFurnace Returns:

True or False

Return typ
boolean

solve_building_lightning (*occupancy_percent : float*) → None

Calculate the lighting of the building for the time step Args:

occupancy_percent: occupancy for the time step

Return type:
None

sys_electricity_folssils_sum (*heating_sys_electricity_hi_sum : int* , *heating_sys_fossils_hi_sum : float*) → float

Calculates sum of heating_sys_electricity_hi_sum and heating_sys_fossils_hi_sum Args:

heating_sys_electricity_hi_sum: heating_sys_fossils_hi_sum:

Returns:
heating_sys_electricity_hi_sum + heating_sys_fossils_hi_sum

Return type

float

wood ( ) → bool

Checks if heating supply system of the building in the list named wood Returns:

True or False

Return typ

boolean

Module contents

dibs_computing_core.iso_simulator.data_source package

Submodules

dibs_computing_core.iso_simulator.data_source.datasource module

class dibs_computing_core.iso_simulator.data_source.datasource. DataSource

Bases: ABC

This interface provides several methods that can be implemented in other classes

abstract choose_and_get_the_right_weather_data_from_path ( ) → list [WeatherData]

This method retrieves the right weather data Args: Returns:

weather_data_objects

abstract get_epw_file ( ) → EPWFile

Function finds the epw file depending on building location, Pick latitude and longitude from plz_data and put values into a list and Calculate minimum distance to next weather station Args:

Returns:

epw_file object

abstract get_epw_pe_factors ( ) → list [PrimaryEnergyAndEmissionFactor]

This method retrieves all primary energy and emission factors Returns:

epw_pe_factors

abstract get_gains ( ) → tuple [tuple [float , str] , float]

Find data from DIN V 18599-10 or SIA2024 Args:

Returns:

gain_person_and_typ_norm, appliance_gains

abstract get_schedule ( ) → tuple [list [ScheduleName] , str , float] | HkOrUkNotFoundError

Find occupancy schedule from SIA2024, depending on hk_geb, uk_geb Args:

Returns:

schedule_name_list, schedule_name or throws an error

abstract get_tek ( ) → tuple [float , str] | HkOrUkNotFoundError

Find TEK values from Partial energy parameters to build the comparative values in accordance with the announcement of 15.04.2021 on the Building Energy Act (GEG) of 2020, depending on hk_geb, uk_geb Args:

Returns:

tek_dhw, tek_name or throws an error

abstract get_usage_time ( ) → tuple [int , int] | UsageTimeError

Find building's usage time DIN 18599-10 or SIA2024 Args:

Returns:

usage_start, usage_end or throws error

abstract get_user_building ( ) → Building

This method retrieves the building to be simulated. Args:

Returns:

building

abstract get_user_buildings ( ) → list [Building]

This method retrieves all the building to be simulated. Args:

Returns:

building

Module contents

dibs_computing_core.iso_simulator.exceptions package

Submodules

dibs_computing_core.iso_simulator.exceptions.building_not_heated_exception module

exception **dibs_computing_core.iso_simulator.exceptions.building_not_heated_exception. BuildingNotHeatedError**

Bases: `Exception`

Raised when the building not heated

dibs_computing_core.iso_simulator.exceptions.ghg_emission module

exception **dibs_computing_core.iso_simulator.exceptions.ghg_emission. GHGEmissionError**

Bases: `Exception`

Raised when an error occured during calculation of GHG-Emission for Heating. The following heating_supply_system cannot be considered yet

dibs_computing_core.iso_simulator.exceptions.plz_exception module

exception **dibs_computing_core.iso_simulator.exceptions.plz_exception. PLZNotFoundError**

Bases: `Exception`

Raised when zipcode not found

dibs_computing_core.iso_simulator.exceptions.uk_or_hk_exception module

exception **dibs_computing_core.iso_simulator.exceptions.uk_or_hk_exception. HkOrUkNotFoundError**

Bases: `Exception`

Raised when hk_geb or uk_geb not found in the dataframe

dibs_computing_core.iso_simulator.exceptions.usage_time_exception module

exception **dibs_computing_core.iso_simulator.exceptions.usage_time_exception. UsageTimeError (** *value : str* **)**

Bases: `Exception`

Raised when something went wrong with the function getUsagetime()

Module contents

dibs_computing_core.iso_simulator.model package

Submodules

dibs_computing_core.iso_simulator.model.ResultOutput module

class dibs_computing_core.iso_simulator.model.ResultOutput. ResultOutput (*building : Building* , *sum_object : CalculationOfSum* , *heating_sys_hi_sum : float* , *heating_sys_electricity_hi_sum : float* , *heating_sys_fossils_hi_sum : float* , *heating_sys_carbon_sum : float* , *heating_sys_pe_sum : float* , *cooling_sys_carbon_sum : float* , *cooling_sys_pe_sum : float* , *hot_water_energy_hi_sum : float* , *heating_fuel_type : str* , *heating_f_ghg : float* , *heating_f_pe : float* , *heating_f_hs_hi : float* , *hot_water_fuel_type : str* , *hot_water_f_ghg : float* , *hot_water_f_pe : float* , *hot_water_f_hs_hi : float* , *cooling_fuel_type : str* , *cooling_f_ghg : float* , *cooling_f_pe : float* , *cooling_f_hs_hi : float* , *light_appl_fuel_type : str* , *light_appl_f_ghg : float* , *light_appl_f_pe : float* , *light_appl_f_hs_hi : float* , *hot_water_sys_carbon_sum : float* , *hot_water_sys_pe_sum : float* , *lighting_demand_carbon_sum : float* , *lighting_demand_pe_sum : float* , *appliance_gains_demand_carbon_sum : float* , *appliance_gains_demand_pe_sum : float* , *carbon_sum : float* , *pe_sum : float* , *fe_hi_sum : float* , *schedule_name : str* , *typ_norm : str* , *epw_filename : str*)

Bases: `object`

calc_appliance_gains_demand_gwp ( )

calc_appliance_gains_demand_pe ( )

calc_cooling_demand ( )

calc_cooling_energy ( )

calc_cooling_sys_electricity ( )

calc_cooling_sys_fossils ( )

calc_cooling_sys_gwp ( )

calc_cooling_sys_pe ( )

calc_electricity_demand_total ( )

calc_electricity_demand_total_ref ( )

calc_fossils_demand_total ( )

calc_fossils_demand_total_ref ( )

calc_gwp ( )

calc_heating_demand ( )

calc_heating_energy ( )

calc_heating_sys_electricity ( )

calc_heating_sys_fossils ( )
calc_heating_sys_gwp ( )
calc_heating_sys_pe ( )
calc_hot_water_demand ( )
calc_hot_water_energy ( )
calc_hot_water_sys_gwp ( )
calc_hot_water_sys_pe ( )
calc_lighting_demand_gwp ( )
calc_lighting_demand_pe ( )
calc_pe ( )

dibs_computing_core.iso_simulator.model.building module

Physics required to calculate sensible space heating and space cooling loads, and space lighting loads (DIN EN ISO 13970:2008)

The equations presented here is this code are derived from ISO 13790 Annex C, Methods are listed in order of apperance in the Annex

Portions of this software are copyright of their respective authors and released under the MIT license: RC_BuildingSimulator, Copyright 2016 Architecture and Building Systems, ETH Zurich

author: "Julian Bischof, Simon Knoll, Michael Hörner " copyright: "Copyright 2023, Institut Wohnen und Umwelt" license: "MIT"

class dibs_computing_core.iso_simulator.model.building. Building (*scr_gebaeude_id : str* , *plz : str* , *hk_geb : str* , *uk_geb : str* , *max_occupancy : int* , *wall_area_og : float* , *wall_area_ug : float* , *window_area_north : float* , *window_area_east : float* , *window_area_south : float* , *window_area_west : float* , *roof_area : float* , *net_room_area : float* , *energy_ref_area : float* , *base_area : float* , *gross_base_area : float* , *building_height : float* , *net_volume : float* , *gross_volume : float* , *envelope_area : float* , *lighting_load : float* , *lighting_control : int* , *lighting_utilisation_factor : float* , *lighting_maintenance_factor : float* , *aw_construction : int* , *shading_device : int* , *shading_solar_transmittance : float* , *glass_solar_transmittance : float* , *glass_solar_shading_transmittance : float* , *glass_light_transmittance : float* , *u_windows : float* , *u_walls : float* , *u_roof : float* , *u_base : float* , *temp_adj_base : float* , *temp_adj_walls_ug : float* , *ach_inf : float* , *ach_win : float* , *ach_vent : float* , *heat_recovery_efficiency : int* , *thermal_capacitance : int* , *t_set_heating : int* , *t_start : int* , *t_set_cooling : int* , *night_flushing_flow : int* , *max_heating_energy_per_floor_area : float* , *max_cooling_energy_per_floor_area : float* , *heating_supply_system : str* , *cooling_supply_system : str* , *heating_emission_system : str* , *cooling_emission_system : str* , *dhw_system*)

Bases: `object`

Sets the parameters of the building.

INPUT PARAMETER DEFINITION scr_gebaeude_id: Building Screening-ID plz: Zipcode of building location hk_geb: Usage type (main category) uk_geb: Usage type (subcategory) max_occupancy: Max. number of persons wall_area_og: Area of all walls above ground in contact with the outside [m2] wall_area_ug: Area of all walls below ground in contact with soil [m2] window_area_north: Area of the glazed surface in contact with the outside facing north [m2] window_area_east: Area of the glazed surface in contact with the outside facing east [m2] window_area_south: Area of the glazed surface in contact with the outside facing south [m2] window_area_west: Area of the glazed surface in contact with the outside facing west [m2] roof_area: Area of the roof in contact with the outside [m2] net_room_area: Area of all floor areas from usable rooms including all floor plan levels of the building (Refers to "Netto-Raumfläche", DIN 277-1:2016-01) base_area: Area for the calculation of transmission heat losses to the soil. Also used to calculate the building's volume. energy_ref_area: Energy reference area of the building building_height: Mean height of the building [m] lighting_load: Lighting Load [W/m2] lighting_control: Lux threshold at which the lights turn on [Lx] lighting_utilisation_factor: A factor that determines how much natural solar lumminace is effectively utilised in the space lighting_maintenance_factor: A factor based on how dirty the windows area glass_solar_transmittance: Solar radiation passing through the window (g-value) glass_solar_shading_transmittance: Solar radiation passing through the window with active shading devices glass_light_transmittance: Solar illuminance passing through the window u_windows: U value of glazed surfaces [W/m2K] u_walls: U value of external walls [W/m2K] u_roof: U value of the roof [W/m2K] u_base: U value of the floor [W/m2K] temp_adj_base: Temperature adjustment factor for the floor temp_adj_walls_ug: Temperature adjustment factor for walls below ground ach_inf: Air changes per hour through infiltration [Air Changes Per Hour] ach_win: Air changes per hour through opened windows [Air Changes Per Hour] ach_vent: Air changes per hour through ventilation [Air Changes Per Hour] ventilation_efficiency: Efficiency of the heat recovery system for ventilation. Set to 0 if there is no heat recovery night_flushing_flow: Air changes per hour through night flushing [Air Changes Per Hour] thermal_capacitance: Thermal capacitance of the building [J/m2K] t_set_heating : Thermal heating set point [C] t_set_cooling: Thermal cooling set point [C] max_cooling_energy_per_floor_area: Maximum cooling load. Set to -np.inf for unrestricted cooling [C] max_heating_energy_per_floor_area: Maximum heating load per floor area. Set to no.inf for unrestricted heating [C] heating_supply_system: The type of heating system cooling_supply_system: The type of cooling system heating_emission_system: How the heat is distributed to the building cooling_emission_system: How the cooling energy is distributed to the building

VARIABLE DEFINITION

internal_gains: Internal Heat Gains [W] solar_gains: Solar Heat Gains after transmitting through the window [W] t_out: Outdoor air temperature [C] t_m_prev: Thermal mass temperature from the previous time step ill: Illuminance transmitting through the window [lumen] occupancy: Occupancy [people] t_m_next: Medium temperature of the next time step [C] t_m: Average between the previous and current time-step of the bulk temperature [C]

Inputs to the 5R1C model: c_m: Thermal Capacitance of the medium [J/K] h_tr_is: Conductance between the air node and the inside surface node [W/K] h_tr_w: Heat transfer coefficient from the outside through windows, doors [W/K] h_tr_op: Heat transfer coefficient from the outside through opaque elements [W/K] h_tr_em: Conductance between outside node and mass node [W/K] h_tr_ms: Conductance between mass node and internal surface node [W/K] h_ve_adj: Ventilation heat transmission coefficient [W/K]

phi_m_tot: see formula for the calculation, eq C.5 in standard [W] phi_m: Combination of internal and solar gains directly to the medium [W] phi_st: combination of internal and solar gains directly to the internal surface [W] phi_ia: combination of internal and solar gains to the air [W] energy_demand: Heating and Cooling of the Supply air [W]

h_tr_1: combined heat conductance, see function for definition [W/K] h_tr_2: combined heat conductance, see function for definition [W/K] h_tr_3: combined heat conductance, see function for definition [W/K]

calc_energy_demand (*internal_gains* , *solar_gains* , *t_out* , *t_m_prev*)

Calculates the energy demand of the space if heating/cooling is active Used in: solve_building_energy() # Step 1 - Step 4 in Section C.4.2 in [C.3 ISO 13790]

calc_energy_demand_unrestricted (*energy_floorAx10* , *t_air_set* , *t_air_0* , *t_air_10*)

Calculates the energy demand of the system if it has no maximum output restrictions # (C.13) in [C.3 ISO 13790]

Based on the Thales Intercept Theorem. Where we set a heating case that is 10x the floor area and determine the temperature as a result Assuming that the relation is linear, one can draw a right angle triangle. From this we can determine the heating level required to achieve the set point temperature This assumes a perfect HVAC control system

calc_h_ve_adj (*hour* , *t_out* , *usage_start* , *usage_end*)

Calculates h_ve_adj depending on the building's usage time

(Eq. 21) in ISO 13790, p. 49-50

Parameters :

- **hour** (*int*) – Hour of the timestep
- **t_out** (*float*) – Outdoor temperature of this timestep
- **usage_start** (*int*) – Beginning of usage time according to SIA2024

- **usage_end** (*int*) – Ending of usage time according to SIA2024

Returns :

self.h_ve_adj

Return type :

float

calc_heat_flow (*t_out* , *internal_gains* , *solar_gains* , *energy_demand*)

Calculates the heat flow from the solar gains, heating/cooling system, and internal gains into the building

The input of the building is split into the air node, surface node, and thermal mass node based on on the following equations

#C.1 - C.3 in [C.3 ISO 13790]

Note that this equation has diverged slightly from the standard as the heating/cooling node can enter any node depending on the emission system selected

calc_phi_m_tot (*t_out*)

Calculates a global heat transfer. This is a definition used to simplify equation calc_t_m_next so it's not so long to write out # (C.5) in [C.3 ISO 13790] # h_ve = h_ve_adj and t_supply = t_out [9.3.2 ISO 13790]

calc_t_air (*t_out*)

Calculate the temperature of the air node # (C.11) in [C.3 ISO 13790] # h_ve = h_ve_adj and t_supply = t_out [9.3.2 ISO 13790]

calc_t_m (*t_m_prev*)

Temperature used for the calculations, average between newly calculated and previous bulk temperature # (C.9) in [C.3 ISO 13790]

calc_t_m_next (*t_m_prev*)

Primary Equation, calculates the temperature of the next time step # (C.4) in [C.3 ISO 13790]

calc_t_s (*t_out*)

Calculate the temperature of the inside room surfaces. Consists of the air temperature and the average radiation temperature # (C.10) in [C.3 ISO 13790] # h_ve = h_ve_adj and t_supply = t_out [9.3.2 ISO 13790] calc_temperatures_crank_nicolson (*energy_demand* , *internal_gains* , *solar_gains* , *t_out* , *t_m_prev*)

Determines node temperatures (t_air, t_m, t_s) and computes derivation to determine the new node temperatures Used in: has_demand(), solve_building_energy(), calc_energy_demand() # section C.3 in [C.3 ISO 13790]

check_night_flushing (*hour* , *t_out*)

Checks if night flushing is on/off

Parameters :

- **hour** (*int*) – Hour of the timestep
- **t_out** (*float*) – Outdoor temperature of this timestep

Returns :

self.night_flushing_on

Return type :

bool

property **h_tr_1**

Definition to simplify calc_phi_m_tot # (C.6) in [C.3 ISO 13790]

property **h_tr_2**

Definition to simplify calc_phi_m_tot # (C.7) in [C.3 ISO 13790]

property **h_tr_3**

Definition to simplify calc_phi_m_tot # (C.8) in [C.3 ISO 13790]

has_demand (*internal_gains* , *solar_gains* , *t_out* , *t_m_prev*)

Determines whether the building requires heating or cooling Used in: solve_building_energy()

step 1 in section C.4.2 in [C.3 ISO 13790]

solve_building_energy (*internal_gains* , *solar_gains* , *t_out* , *t_m_prev*)

Calculates the heating and cooling consumption of a building for a set timestep

Parameters :

- **internal_gains** (*float*) – internal heat gains from people and appliances [W]

- **solar_gains** (*float*) – solar heat gains [W]
- **t_out** (*float*) – Outdoor air temperature [C]
- **t_m_prev** (*float*) – Previous air temperature [C]

Returns :

self.heating_demand, space heating demand of the building

Returns :

self.heating_sys_electricity, heating electricity consumption

Returns :

self.heating_sys_fossils, heating fossil fuel consumption

Returns :

self.cooling_demand, space cooling demand of the building

Returns :

self.cooling_sys_electricity, electricity consumption from cooling

Returns :

self.cooling_sys_fossils, fossil fuel consumption from cooling

Returns :

self.electricity_out, electricity produced from combined heat pump systems

Returns :

self.sys_total_energy, total exergy consumed (electricity + fossils) for heating and cooling

Returns :

self.heating_energy, total exergy consumed (electricity + fossils) for heating

Returns :

self.cooling_energy, total exergy consumed (electricity + fossils) for cooling

Returns :

self.cop, Coefficient of Performance of the heating or cooling system

Return type :

float

solve_building_lighting (*illuminance* , *occupancy*)

Calculates the lighting demand for a set timestep

Daylighting is based on methods in Szokolay, S.V. (1980): Environmental Science Handbook vor architects and builders. Unknown Edition, The Construction Press, Lancaster/London/New York, ISBN: 0-86095-813-2, p. 105ff. respectively Szokolay, S.V. (2008): Introduction to Architectural Science. The Basis of Sustainable Design. 2nd Edition, Elsevier/Architectural Press, Oxford, ISBN: 978-0-7506-8704-1, p. 154ff.

Parameters :

- **illuminance** (*float*) – Illuminance transmitted through the window [Lumens]
- **occupancy** (*float*) – Probability of full occupancy

Returns :

self.lighting_demand, Lighting Energy Required for the timestep

Return type :

float

property **t_opperative**

The opperative temperature is a weighted average of the air and mean radiant temperatures. It is not used in any further calculation at this stage # (C.12) in [C.3 ISO 13790]

dibs_computing_core.iso_simulator.model.calculations_sum module

This class is used to store the sum of all the simulated hours for a building

class dibs_computing_core.iso_simulator.model.calculations_sum. CalculationOfSum (*HeatingDemand_sum : float* , *HeatingEnergy_sum : float* , *Heating_Sys_Electricity_sum : float* , *Heating_Sys_Fossils_sum : float* , *CoolingDemand_sum : float* , *CoolingEnergy_sum : float* , *Cooling_Sys_Electricity_sum : float* , *Cooling_Sys_Fossils_sum : float* , *HotWaterDemand_sum : float* , *HotWaterEnergy_sum : float* , *HotWater_Sys_Electricity_sum : float* , *HotWater_Sys_Fossils_sum : float* , *InternalGains_sum : float* , *Appliance_gains_demand_sum : float* , *Appliance_gains_elt_demand_sum : float* , *LightingDemand_sum : float* , *SolarGainsSouthWindow_sum : float* , *SolarGainsEastWindow_sum : float* , *SolarGainsWestWindow_sum : float* , *SolarGainsNorthWindow_sum : float* , *SolarGainsTotal_sum : float*)

Bases: `object`

dibs_computing_core.iso_simulator.model.epw_file module

class dibs_computing_core.iso_simulator.model.epw_file. EPWFile (*file_name : str* , *coordinates_station : List* , *distance : float*)

Bases: `object`

Sets the parameters of the EPWFile.

Args:

file_name: The name of the EPW file coordinates_station: The coordinates of the station distance: The nearest distance between the station and the building

dibs_computing_core.iso_simulator.model.hours_result module

class dibs_computing_core.iso_simulator.model.hours_result. Result

Bases: `object`

This class contains the result for the calculation of a building

Args:

heating_demand (List[float]): Space heating demand of the building heating_energy (List[float]): Total exergy consumed (electricity + fossils) for heating heating_sys_electricity (List[float]): Hating electricity consumption heating_sys_fossils (List[float]): Hating fossil fuel consumption cooling_demand (List[float]): Space cooling demand of the building cooling_energy (List[float]): Total exergy consumed (electricity + fossils) for cooling cooling_sys_electricity (List[float]): Electricity consumption from cooling cooling_sys_fossils (List[float]): Fossil fuel consumption from cooling hotwater_demand (List[float]): Hot water demand of the building hotwater_energy (List[float]): hotwater_sys_electricity (List[float]): hotwater_sys_fossils (List[float]): temp_air (List[float]): Temperature of the air outside_temp (List[float]): Temperature of the outside air lighting_demand (List[float]): Lighting Energy Required for the timestep internal_gains (List[float]): Internal Heat Gains [W] solar_gains_south_window (List[float]): solar_gains_east_window (List[float]): solar_gains_west_window (List[float]): solar_gains_north_window (List[float]): solar_gains_total (List[float]): day_time: hotwaterdemand: hotwaterenergy: hot)water_sys_electricity: hot_water_sys_fossils:

append_results (*building : Building* , *all_windows : list [Window]* , *hot_water_demand : float* , *hot_water_energy : float* , *hot_water_sys_electricity : float* , *hot_water_sys_fossils : float* , *t_out : float* , *internal_gains : float* , *appliance_gains_demand : float* , *appliance_gains_elt_demand : float* , *solar_gains_all_windows : float* , *hour : int*) → None

This method appends the result of a simulated hour in the object result Args:

building: all_windows: hot_water_demand: hot_water_energy: hot_water_sys_electricity: hot_water_sys_fossils: t_out: internal_gains: appliance_gains_demand: appliance_gains_elt_demand: solar_gains_all_windows: hour:

Returns:

None

calc_sum_of_results ( ) → CalculationOfSum

Calculates the sum of results Returns:

sum_of_all_results

dibs_computing_core.iso_simulator.model.location module

class dibs_computing_core.iso_simulator.model.location. Location

Bases: `object`

calc_sun_position (*latitude_deg* , *longitude_deg* , *year* , *hoy*)

Calculates the sun position for a specific hour and location

Parameters :

- **latitude_deg** (*float*) – Geographical Latitude in Degrees
- **longitude_deg** (*float*) – Geographical Longitude in Degrees
- **year** (*int*) – year
- **hoy** (*int*) – Hour of the year from the start. The first hour of January is 1

Returns :

altitude, azimuth: Sun position in altitude and azimuth degrees [degrees]

Return type :

tuple

dibs_computing_core.iso_simulator.model.primary_energy_and_emission_factors module

class dibs_computing_core.iso_simulator.model.primary_energy_and_emission_factors. PrimaryEnergyAndEmissionFactor (*energy_carrier : str* , *primary_energy_factor_GEG : float* , *relation_calorific_to_heating_value_GEG : float* , *gwp_spezific_to_heating_value_GEG : int* , *use : str*)

Bases: `object`

dibs_computing_core.iso_simulator.model.schedule_name module

class dibs_computing_core.iso_simulator.model.schedule_name. ScheduleName (*People : float* , *Appliances : float*)

Bases: `object`

dibs_computing_core.iso_simulator.model.summary_result module

class dibs_computing_core.iso_simulator.model.summary_result. SummaryResult (*result : ResultOutput* , *user_args : list*)

Bases: `object`

dibs_computing_core.iso_simulator.model.weather_data module

class dibs_computing_core.iso_simulator.model.weather_data. WeatherData (*year* , *month* , *day* , *hour* , *minute* , *datasource* , *drybulb_C* , *dewpoint_C* , *relhum_percent* , *atmos_Pa* , *exthorrad_Whm2* , *extdirrad_Whm2* , *horirsky_Whm2* , *glohorrad_Whm2* , *dirnorrad_Whm2* , *difhorrad_Whm2* , *glohorillum_lux* , *dirnorillum_lux* , *difhorillum_lux* , *zenlum_lux* , *winddir_deg* , *windspd_ms* , *totskycvr_tenths* , *opaqskycvr_tenths* , *visibility_km* , *ceiling_hgt_m* , *presweathobs* , *presweathcodes* , *precip_wtr_mm* , *aerosol_opt_thousandths* , *snowdepth_cm* , *days_last_snow* , *Albedo* , *liq_precip_depth_mm* , *liq_precip_rate_Hour*)

Bases: `object`

dibs_computing_core.iso_simulator.model.window module

class dibs_computing_core.iso_simulator.model.window. Window (*azimuth_tilt* , *alititude_tilt = 90* , *glass_solar_transmittance = 0.7* , *glass_solar_shading_transmittance = 0.2* , *glass_light_transmittance = 0.8* , *area = 1*)

Bases: `object`

calc_diffuse_solar_factor ( ) → float

Calculates the proportion of diffuse radiation

calc_direct_solar_factor (*sun_altitude : float* , *sun_azimuth : float*) → float

Calculates the cosine of the angle of incidence on the window

calc_illuminance (*sun_altitude : float* , *sun_azimuth : float* , *normal_direct_illuminance : int* , *horizontal_diffuse_illuminance : int*) → None

Calculates the illuminance in the building zone through the set window

Parameters :

- **sun_altitude** (*float*) – Altitude Angle of the Sun in Degrees

- **sun_azimuth** (*float*) – Azimuth angle of the sun in degrees
- **normal_direct_illuminance** (*float*) – Normal Direct Illuminance from weather file [Lx]
- **horizontal_diffuse_illuminance** (*float*) – Horizontal Diffuse Illuminance from weather file [Lx]

Returns :

self.incident_illuminance, Incident Illuminance on window [Lumens]

Returns :

self.transmitted_illuminance - Illuminance in building after transmitting through the window [Lumens]

Return type :

float

calc_solar_gains (*sun_altitude : float* , *sun_azimuth : float* , *normal_direct_radiation : int* , *horizontal_diffuse_radiation : int* , *t_air : float* , *hour : int*) → None

Calculates the solar gains in the building zone through the set window

Parameters :

- **sun_altitude** (*float*) – Altitude Angle of the Sun in Degrees
- **sun_azimuth** (*float*) – Azimuth angle of the sun in degrees
- **normal_direct_radiation** (*float*) – Normal Direct Radiation from weather file
- **horizontal_diffuse_radiation** (*float*) – Horizontal Diffuse Radiation from weather file

Added: #param t_out: Outdoor temperature from weather file :type t_out: float

Returns :

self.incident_solar, Incident Solar Radiation on window

Returns :

self.solar_gains - Solar gains in building after transmitting through the window

Return type :

float

set_variable_for_calc_sun (*sun_altitude : float* , *sun_azimuth : float* , *normal_direct_radiation : int* , *horizontal_diffuse_radiation : int*) → None

Module contents

Submodules

dibs_computing_core.iso_simulator.emission_system module

Emission System Parameters for Heating and Cooling

Model of different Emission systems. New Emission Systems can be introduced by adding new classes

Temperatures only relevant in combination with heat pumps at this stage Temperatures taken from RC_BuildingSimulator and CEA (https://github.com/architecture-building-systems/CityEnergyAnalyst/blob/master/cea/databases/CH/assemblies/HVAC.xls)

Portions of this software are copyright of their respective authors and released under the MIT license: RC_BuildingSimulator, Copyright 2016 Architecture and Building Systems, ETH Zurich

author: “Simon Knoll, Julian Bischof, Michael Hörner ” copyright: “Copyright 2021, Institut Wohnen und Umwelt” license: “MIT”

class **dibs_computing_core.iso_simulator.emission_system. AirConditioning** (*energy_demand*)

Bases: `EmissionSystemBase`

All heat is given to the air via an AC-unit. HC input via the air node as in the ISO 13790 Annex C Temperatures taken from RC_BuildingSimulator [new radiators (assumption)] Heat is emitted to the air node heat_flows ( )

class dibs_computing_core.iso_simulator.emission_system. EmissionDirector

Bases: `object`

The director sets what Emission system is being used, and runs that set Emission system

builder = *None*

calc_flows ( )

set_builder (*builder*)

class dibs_computing_core.iso_simulator.emission_system. EmissionSystemBase (*energy_demand*)

Bases: `object`

The base class in which systems are built from

heat_flows ( )

class dibs_computing_core.iso_simulator.emission_system. Flows

Bases: `object`

A base object to store output variables

cooling_supply_temperature = *nan*

heating_supply_temperature = *nan*

phi_ia_plus = *nan*

phi_m_plus = *nan*

phi_st_plus = *nan*

class dibs_computing_core.iso_simulator.emission_system. NoCooling (*energy_demand*)

Bases: `EmissionSystemBase`

Dummy Class used for buildings with no cooling supply system

heat_flows ( )

class dibs_computing_core.iso_simulator.emission_system. NoHeating (*energy_demand*)

Bases: `EmissionSystemBase`

Dummy Class used for buildings with no heating supply system

heat_flows ( )

class dibs_computing_core.iso_simulator.emission_system. SurfaceHeatingCooling (*energy_demand*)

Bases: `EmissionSystemBase`

All HC energy goes into the surface node, assumed low supply temperature Heat is emitted to the surface node

heat_flows ( )

class dibs_computing_core.iso_simulator.emission_system. ThermallyActivated (*energy_demand*)

Bases: `EmissionSystemBase`

Heat is emitted to the thermal mass node, assumed low supply temperature

heat_flows ( )

dibs_computing_core.iso_simulator.supply_system module

Supply System Parameters for Heating and Cooling

Model of different Supply systems. New Supply Systems can be introduced by adding new classes

TODO: Have a look at CEA calculation methodology https://github.com/architecture-building-systems/CEAforArcGIS/blob/master/cea/technologies/heatpumps.py

Portions of this software are copyright of their respective authors and released under the MIT license: RC_BuildingSimulator, Copyright 2016 Architecture and Building Systems, ETH Zurich

author: "Simon Knoll, Julian Bischof, Michael Hörner " copyright: "Copyright 2021, Institut Wohnen und Umwelt" license: "MIT"

class dibs_computing_core.iso_simulator.supply_system. AbsorptionRefrigerationSystem (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Wärmeabfuhr Kältemaschine (Kondensator): Wassergekühlt (Primärkreis) Verdichterart: Absorptionskälteanlage H2O/LiBr

Assumption: Driving heat comes from waste heat, not from fossils (this may lead to biased results if this is not the case), due to the fact that absorption chillers usually have a lower efficiency compared to compression chillers. We assume that building owners only use absorption chillers if they have access to heat free of charge.

Furthermore: Electricity consumption for pumps etc. are not considered at this stage

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. AirCooledPistonScroll (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Wärmeabfuhr Kältemaschine (Kondensator): Luftgekühlt (Primärkreis) Verdichterart: Kolben-/Scrollverdichter - on/off Betrieb

Informationsblatt zur Kälteerzeugung gemäss Norm SIA 382-1:2014, S. 4 Kälteerzeugerleistung der Kältemaschine: 100 kW EER (full load): 3,1

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. AirCooledPistonScrollMulti (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Wärmeabfuhr Kältemaschine (Kondensator): Luftgekühlt (Primärkreis) Verdichterart: Kolben-/Scrollverdichter - mehrstufig

Informationsblatt zur Kälteerzeugung gemäss Norm SIA 382-1:2014 Kälteerzeugerleistung der Kältemaschine: 100 kW EER (full load): 3,1

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. BiogasBoilerCondensingBefore95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel vor 1995 - Biogas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. BiogasBoilerCondensingFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel nach 1995 - Biogas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. BiogasOilBoilerCondensingFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel ab 1995 - Biogas/Bioöl Mix

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. BiogasOilBoilerCondensingImproved (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel verbessert - Biogas/Bioöl Mix

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. BiogasOilBoilerLowTempBefore95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel vor 1995 - Biogas/Bioöl Mix

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. CoalSolidFuelBoiler (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Mix between Feststoffkessel 78-94 (Kohle) - Steinkohle and Feststoffkessel 78-94 (Kohle) - Braunkohle

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. DirectCooler (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Created by PJ to check accuracy against previous simulation

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. DirectHeater (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Created by PJ to check accuracy against previous simulation

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. DistrictCooling (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

DistrictCooling assumed with efficiency 100%

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. DistrictHeating (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 District Heating with expenditure factor = 1.002

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. ElectricHeating (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Straight forward electric heating. 100 percent conversion to heat.

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerCondensingBefore95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel vor 1995 - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerCondensingFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel ab 1995 - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerCondensingImproved (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel verbessert - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerLowTempBefore87 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel vor 1987 - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerLowTempBefore95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel vor 1995 - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerLowTempFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel ab 1995 - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerLowTempSpecialFrom78 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel-Spezialkessel ab 1978 - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerLowTempSpecialFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel-Spezialkessel ab 1995 - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerStandardBefore86 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Konstanttemperaturkessel vor 1986 - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerStandardBefore95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Konstanttemperaturkessel vor 1995 (87-94) - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasBoilerStandardFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Konstanttemperaturkessel ab 1995 - Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasCHP (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Combined heat and power unit with 49 percent thermal and 38 percent electrical efficiency. Source: Arbeitsgemeinschaft für sparsamen und umwelfreundlichen Energieverbrauch e.V. (2011): BHKW-Kenndasten 2011

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. GasEnginePistonScroll (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

ANALYSIS OF ENERGY EFFICIENCY OF GAS DRIVEN HEAT PUMPS - PhD Work of M.Sc. Essam Mahrous Elgenady Elgendy Fakultaet fuer Verfahrens- und Systemtechnik der Otto-von-Guericke-Universitaet Magdeburg

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. HeatPumpAirSource (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

BETA Version COP based off regression analysis of manufacturers data Source: Staffell et al. (2012): A review of domestic heat pumps, In: Energy & Environmental Science, 2012, 5, p. 9291-9306

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. HeatPumpGroundSource (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

" BETA Version Ground source heat pumps can be designed in an open-loop system where they "extract water directly from, and reject it back to rivers or groundwater resources such as aquifers and springs" or in an closed-loop system where they use "a sealed loop to extract heat from the surrounding soil or rock". Source: Staffell et al. (2012): A review of domestic heat pumps, In: Energy & Environmental Science, 2012, 5, p. 9291-9306

Reservoir temperatures 7 degC (winter) and 12 degC (summer). COP based on regression analysis of manufacturers data Source: Staffell et al. (2012): A review of domestic heat pumps, In: Energy & Environmental Science, 2012, 5, p. 9291-9306

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. LGasBoilerCondensingBefore95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel vor 1995 - L-Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. LGasBoilerCondensingFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel ab 1995 - L-Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. LGasBoilerCondensingImproved (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel verbessert - L-Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. LGasBoilerLowTempBefore87 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel vor 1987 - L-Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. LGasBoilerLowTempBefore95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel vor 1995 - L-Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. LGasBoilerLowTempFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel ab 1995 - L-Gas

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. NoCooler (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Dummyclass used for buildings with no cooling supply system

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. NoHeater (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Dummyclass used for buildings with no heating supply system

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. OilBoilerCondensingBefore95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel vor 1995 - Oil

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. OilBoilerCondensingFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel ab 1995 - Oil

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. OilBoilerCondensingImproved (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Brennwertkessel verbessert

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. OilBoilerLowTempBefore87 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel vor 1987 - Oil

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. OilBoilerLowTempBefore95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel vor 1995 - Oil

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. OilBoilerLowTempFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Niedertemperaturkessel ab 1995 - Oil

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. OilBoilerStandardBefore86 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Konstanttemperaturkessel 78-86 - Oil

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. OilBoilerStandardFrom95 (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Konstanttemperaturkessel ab 1995 - Oil

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. SolidFuelLiquidFuelFurnace (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Minimum efficiency according to '1. BImSchV, Anlage 4'

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. SupplyDirector

Bases: `object`

The director sets what Supply system is being used, and runs that set Supply system

builder = *None*

calc_system ( )

set_builder (*builder*)

class dibs_computing_core.iso_simulator.supply_system. SupplyOut

Bases: `object`

The System class which is used to output the final results

cop = *nan*

electricity_in = *nan*

electricity_out = *nan*

fossils_in = *nan*

class dibs_computing_core.iso_simulator.supply_system. SupplySystemBase (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `object`

The base class in which Supply systems are built from

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. WaterCooledPistonScroll (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

Wärmeabfuhr Kältemaschine (Kondensator): Wassergekühlt (Primärkreis) Verdichterart: Kolben-/Scrollverdichter - on/off Betrieb

Informationsblatt zur Kälteerzeugung gemäss Norm SIA 382-1:2014 Kälteerzeugerleistung der Kältemaschine: 100 kW EER (full load): 4,25

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. WoodChipSolidFuelBoiler (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Feststoffkessel mit Pufferspeicher ab 95 (Holzhack)

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. WoodPelletSolidFuelBoiler (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Feststoffkessel mit Pufferspeicher ab 95 (Holzpellet)

calc_loads ( )

class dibs_computing_core.iso_simulator.supply_system. WoodSolidFuelBoilerCentral (*load* , *t_out* , *heating_supply_temperature* , *cooling_supply_temperature* , *has_heating_demand* , *has_cooling_demand*)

Bases: `SupplySystemBase`

expenditure factor (=Erzeugeraufwandszahl) from TEK-Tool 9.24 Feststoffkessel ab (ohne Puffer) 95 (Holzhack/Pellets Mix)

calc_loads ( )

Module contents

Module contents

Indices and tables

© Copyright 2024, IWUGERMANY.
Created using Sphinx 7.2.6.