Konfigurator für OSM-Datenaufbereitungs-Prozesse

Release 1.0.0

Felix Weik, Jan-Phillip Hansen, Karl Bernhard, Pascal Dawideit, Si

Jan 12, 2023

USER GUIDE

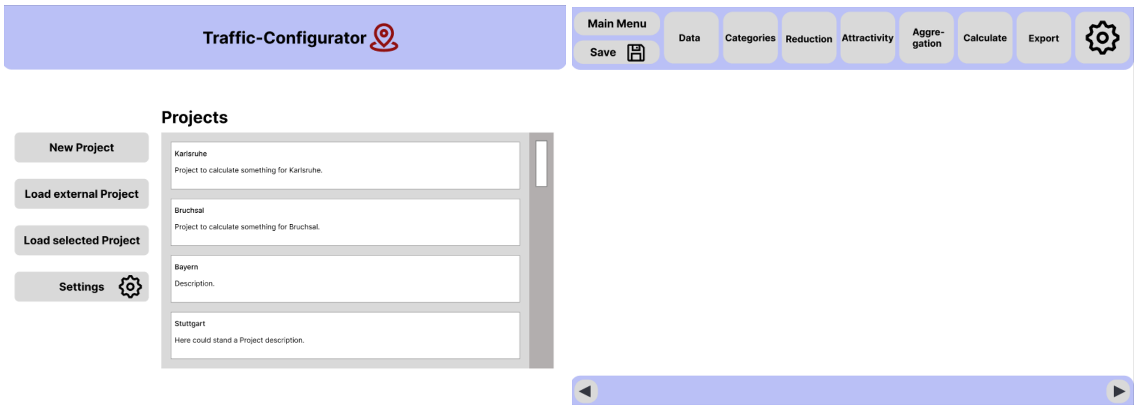
CHAPTER

ONE

KONFIGURATORFUEROSMDATEN

license MIT

Whether it's biking to college or driving to the supermarket, traffic affects us all. all. This product generates from geodata of the free project 'OpenStreetMap' (OSM) a numerical ranking of the attractiveness of geographic locations. A main focus is the configurability of the generation of this score. Traffic planners can easily generate the right data for their traffic forecasting models.



Karlsruhe
Project to calculate something for Karlsruhe.

Bruchsal
Project to calculate something for Bruchsal.

Bayern
Description.

Stuttgart
Here could stand a Project description.

This project was developed in the context of the lecture "Praxis der Softwareentwicklung(PSE)" at the University KIT in 2022,the waterfall model with feedback was used to develop the project. For each of this phase you can find the documentation in the corresponding subfolder, in the github , most of the documentation is in german.

For a more specific description of the code, check the `README.md` in the subfolder *pythoncode*.

1.1 Installation

We recommend installing KonfiguratorFuerOSMDaten using one of the available built distributions, for example using `pip` or `conda`:

```
$ here will stand sth. in the future
```

1.2 Usage

To use the application after installing it, simply run the program. For an description about how the application works check out the folder `Pflichtenheft`.

1.3 License

KonfiguratorFuerOSMDaten is licensed under the MIT License.

1.4 Contribution

To contribute to the project:

1. Fork it (https://github.com/LuposX/KonfiguratorFuerOSMDaten/fork)
2. Create your feature branch (`git checkout -b feature/fooBar`)
3. Commit your changes (`git commit -am 'Add some fooBar'`)
4. Push to the branch (`git push origin feature/fooBar`)
5. Create a new Pull Request

CHAPTER

TWO

USER MANUAL

2.1 Installation

2.1.1 Installation for local development

To install KonfiguratorFuerOSMDaten for local devlopment, clone the package from Github:

```
$ git clone https://github.com/LuposX/KonfiguratorFuerOSMDaten.git
$ cd KonfiguratorFuerOSMDaten
```

For development, use of a virtual environment is strongly recommended. For example using `venv`:

```
$ python3 -m PSE .
$ pip install -r requirements.txt
(PSE) $
```

Or using conda:

```
$ conda env create --file enviroment2.yml
$ conda activate PSE
(PSE) $
```

2.1.2 Testing KonfiguratorFuerOSMDaten

To test hat the installed libaries are working correctly the tests in the `test` folder can be used:

```
$ cd pythoncode
$ cd tests
$ cd libaryTests
```

Each file is a test for one functionality of a libary that is needed in our project, to test `.py` files:

```
$ python script_name.py
```

To test `.ipynb` files:

```
$ jupyter-lab
```

and run the files in jupyter-lab. If the libaries are correctly installed you shouldn't get any errors, when running a file.

2.2 The KonfiguratorFuerOSMDaten Manual

Author
: Simon

Version
: 1.0

Date
: Jan 12, 2023

Copyright
: This work is licensed under a *MIT* license.

Abstract
: This document explains how to use the KonfiguratorFuerOSMDaten Application.

2.2.1 Introduction

Note: Here will stand sth. in the future

For an description about how the application works check out the folder `Pflichtenheft`.

CHAPTER

THREE

CLASS DESCRIPTIONS

3.1 src

3.1.1 src package

Subpackages

src.osm_configurator package

Subpackages

src.osm_configurator.control package

Submodules

src.osm_configurator.control.aggregation_controller module

`class AggregationController`(*model*)

Bases: `object`

The AggregationController is responsible for consistently forwarding requests to the model, regarding the aggregation-calculations and the aggregation methods of the currently selected project.

`__init__`(*model*)

Creates a new instance of the AggregationController, with an association to the model.

Parameters

`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

`get_aggregation_methods`()

Returns a list of all aggregation methods that are available. This function returns all available aggregation methods, not just the ones that are active in the current project.

Returns

The list of the available aggregation methods

Return type

list[*aggregation_method_enum.AggregationMethod*]

is_aggregation_method_active(*method*)

Checks, whether an aggregation method is active in the currently selected project.

Parameters

method (aggregation_method_enum.AggregationMethod) – The aggregation method that is checked for.

Returns

True, if there is currently a project selected and the given aggregation method is active in it; False otherwise.

Return type

bool

set_aggregation_method_active(*method*, *active*)

Activates or deactivates an aggregation method (of the currently selected project). Activates the given method, if active=True and deactivates it otherwise.

Parameters

- **method** (aggregation_method_enum.AggregationMethod) – The aggregation method we want to deactivate/activate
- **active** (*bool*) – True, if we want to activate the given method; False, if we want to deactivate it.

Returns

True, if a project is currently selected and the aggregation method was (de-)activated successfully; False, otherwise.

Return type

bool

src.osm_configurator.control.application_controller module

class ApplicationController

Bases: object

The application controller is responsible for creating the model, the view and the control. It is the start of the application and boots everything up.

main()

Starts the application. This class method's only job is, to give control to an instance of the ApplicationController.

__init__()

Creates a new Application. It creates the view, the model and the control. It is responsible for starting everything up and to switch to the normal workflow of the application.

src.osm_configurator.control.calculation_controller module

`class CalculationController`(*model*)

Bases: `object`

The CalculationController is responsible for forwarding requests to the model, regarding calculations. It may be used to gather information and to control the calculation-process of the currently selected project.

`__init__`(*model*)

Creates a new instance of the CalculationController, with an association to the model.

Parameters
`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

`start_calculations`(*starting_phase*)

Starts the calculations in the given calculation phase in the currently selected project. The calculation process is split in different calculation phases. This function starts the calculation in a given phase.

Parameters
`starting_phase` (`calculation_phase_enum.CalculationPhase`) – The phase, in which the calculation should start

Returns
The status of the calculation: RUNNING, if the calculation was started successfully. For details on the meaning of this return value, see CalculationState

Return type
calculation_state_enum.CalculationState

`get_calculation_state`()

Gives the current calculation state of the selected project.

Returns
Returns the current state of the calculation. For details see documentation of CalculationState.

Return type
calculation_state_enum.CalculationState

`get_current_calculation_phase`()

Returns the calculation phase of the currently selected project.

Returns
The phase, that is currently running. NONE, if no phase is currently running.

Return type
calculation_phase_enum.CalculationPhase

`get_current_calculation_process`()

Returns an approximation of the progress of the calculations in the currently selected project. The progress is given as a number between 0 and 1, where 0 indicates that the calculation has not started yet and 1 indicates, that the calculations are done.

Returns
The value of the approximation.

Return type
float

`cancel_calculations()`

Cancels the calculations of the currently selected project. The calculation phase that is currently running will be stopped.

Returns
True, if the calculation was canceled successfully; False, otherwise.

Return type
bool

src.osm_configurator.control.category_controller module

`class CategoryController(model)`

Bases: `object`

The CategoryController is responsible for consistently forwarding requests to the model, regarding changes to the categories of the current project.

`__init__(model)`

Creates a new instance of the CategoryController, with an association to the model.

Parameters
`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

`check_conflicts_in_category_configuration(path)`

Checks for a given file, if it is a valid category-file and checks, whether there are naming conflicts with the categories of the currently selected project.

Parameters
`path` (*`pathlib.Path`*) – The path to the category-file

Returns
True, if there is currently a project selected and there are no naming conflicts; False, otherwise.

Return type
bool

`import_category_configuration(path)`

Imports the given categories into the currently selected project. Adds the given categories to the category list of the project.

Parameters
`path` (*`pathlib.Path`*) – The path to the category file

Returns
True, if the categories where added successfully; False, if there is no project loaded, the category file is corrupted or the category file does not exist.

Return type
bool

`get_list_of_categories()`

Returns the list of all categories, that are currently in the currently selected project.

Returns
A list of the categories of the project in no particular order.

Return type
list[*category.Category*]

`create_category()`

Creates a new category in the currently selected project. A new category is added to the list of categories of the project. The category has empty properties, except for an arbitrary name. If the creation fails, none will be returned and there won't be a category added.

Returns
The newly created category, none if there was an error

Return type
category.Category

`delete_category(`*category*`)`

Deletes the given category. Removes the given category from the list of categories of the currently selected project

Parameters
category (`category.Category`) – The category, to be deleted

Returns
True, if the category was deleted successfully; False, otherwise

Return type
bool

`get_list_of_key_recommendations(`*current_input*`)`

Returns a list of recommended keys, based on the input that is already entered by the user.

Parameters
current_input (*str*) – The input, that is currently written by the user.

Returns
A list of key recommendations, based on the current_input.

Return type
list[str]

`get_attractivities_of_category(`*category*`)`

Returns the attractivity attributes that are defined for the given category.

Parameters
category (`category.Category`) – The category, whose attractivities are of interest.

Returns
The list of attractivity attributes of the given category

Return type
list[*attractivity_attribute.AttractivityAttribute*]

src.osm_configurator.control.control module

`class Control`

Bases: *IControl*

This class provides a consistent interface for access to the control-package. It is a facade, to make access easy. The control manages the access to the module. That's why this interface should give access to all the features provided by the model.

This implementation of the interface IControl forwards all requests to other classes of this package. For details see the documentation of the corresponding functions.

`__init__()`

Creates a new instance of Control, with a association to the model.

Parameters
`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

`get_list_of_passive_projects()`

Returns the list of (passive) projects, which are in the default project folder of the application.

Returns
The list of passive projects in the default project folder.

Return type
list[*passive_project.PassiveProject*]

`load_project`(*path*)

Loads a project All relevant data of a project are verified and loaded in memory. All coming project-referring calls will be directed to the given project.

Parameters
`path` (*`pathlib.Path`*) – The path to the project folder of the project, to be loaded.

Returns
True, if the project was loaded successfully; False if an error occurred, while trying to load the project. An error happens, if the path is not pointing to a valid project folder or if the project has corrupted files.

Return type
bool

`create_project`(*name*, *description*, *destination*)

Creates a new project with the given attributes and loads it. The model creates a new project folder at the given destination, all relevant files are generated and the project is loaded into memory.

Parameters

- `name` (*`str`*) – The name of the to-be-created project, may not contain any line-breaks.
- `description` (*`str`*) – The description of the to-be-created project. May contain line-breaks.
- `destination` (*`pathlib.Path`*) – The path to the location, where the project-folder of the project should be created.

Returns
True, if the project was created successfully; False if an error occurred. An error occurs, if the name of the project is not valid, if the destination-path is not valid or if the destination-location is already occupied.

Return type
bool

`delete_passive_project`(*project*)

Deletes a project out of the default project folder.

Parameters
`project` (`passive_project.PassiveProject`) – The project, that is going to be deleted.

Returns
True, if the (passive) project has been deleted successfully; False otherwise: The project does not exist or the application has not the right permissions to delete the project.

Return type
bool

`save_project()`

Saves the project. The currently selected project is stored on the disk. All progress made since the last saving are saved.

Returns
True, if the project was saved successfully; False if an error occurred, while attempting to save the project or when there is no project selected.

Return type
bool

`set_current_config_phase(`*config_phase*`)`

Stores the current configuration phase in the model.

Parameters
config_phase (`config_phase_enum.ConfigPhase`) – The new configuration phase.

Returns
True, if setting the configuration phase was successful; False, otherwise.

Return type
bool

`get_current_config_phase()`

Returns the configuration phase, that is currently stored in the model.

Returns
The configuration phase, that is currently stored in the model.

Return type
config_phase_enum.ConfigPhase

`is_project_loaded()`

Checks, whether any project is currently loaded/selected.

Returns
True, if a project is currently selected; False, otherwise.

Return type
bool

`set_osm_data_reference(`*path*`)`

Sets the reference to the osm-data for the selected project. The reference contains the osm-data used in the calculations of the project. This method does not check if the given data is valid.

Parameters
path (*`pathlib.Path`*) – The reference to the osm-data

Returns
True, if the new reference was set successfully; False, if an error occurred while setting the reference.

Return type
bool

`get_osm_data_reference()`

Returns the path to the osm-data, that is used in the currently selected project.

Returns
The path to the osm-data of the currently selected project.

Return type
pathlib.Path

`get_cut_out_mode()`

Gets the method of how the geofilter shall cut out on the OSM-Data.

Returns
The cut-out-mode of the currently selected project.

Return type
cut_out_mode_enum.CutOutMode

`set_cut_out_mode`(*mode*)

Sets the method of how the geofilter shall cut out on the OSM-Data.

Parameters
mode (`cut_out_mode_enum.CutOutMode`) – The mode, to be set

Returns
True, if the CutOutMode was set successfully; False, if an error occurred or no project is currently selected.

Return type
bool

`set_cut_out_reference`(*path*)

Sets the reference to the cut-out file of the currently selected project. This file is later used to calculate the geofilter.

Parameters
path (*pathlib.Path*) – The path to the file containing the cut-out-geometries

Returns
True, if the reference was set successfully; False, if an error occurred. An error occurs, if no project is currently selected or if the given path is not valid or occupied.

Return type
bool

`get_cut_out_reference()`

Gets the reference to the cut-out file of the currently selected project.

Returns
The current reference to the cut-out file.

Return type
pathlib.Path

`check_conflicts_in_category_configuration`(*path*)

Checks for a given file, if it is a valid category-file and checks, whether there are naming conflicts with the categories of the currently selected project.

Parameters
path (*pathlib.Path*) – The path to the category-file

Returns
True, if there is currently a project selected and there are no naming conflicts; False, otherwise.

Return type
bool

`import_category_configuration`(*path*)

Imports the given categories into the currently selected project. Adds the given categories to the category list of the project.

Parameters
`path` (*pathlib.Path*) – The path to the category file

Returns
True, if the categories where added successfully; False, if there is no project loaded, the category file is corrupted or the category file does not exist.

Return type
bool

`get_list_of_categories`()

Returns the list of all categories, that are currently in the currently selected project.

Returns
A list of the categories of the project in no particular order.

Return type
list[*category.Category*]

`create_category`()

Creates a new category in the currently selected project. A new category is added to the list of categories of the project. The category has empty properties, except for an arbitrary name. If the creation fails, none will be returned and there won't be a category added.

Returns
The newly created category, none if there was an error

Return type
category.Category

`delete_category`(*category*)

Deletes the given category. Removes the given category from the list of categories of the currently selected project

Parameters
`category` (`category.Category`) – The category, to be deleted

Returns
True, if the category was deleted successfully; False, otherwise

Return type
bool

`get_list_of_key_recommendations`(*current_input*)

Returns a list of recommended keys, based on the input that is already entered by the user.

Parameters
`current_input` (*str*) – The input, that is currently written by the user.

Returns
A list of key recommendations, based on the current_input.

Return type
list[str]

`get_attractivities_of_category`(*category*)

Returns the attractivity attributes that are defined for the given category.

Parameters
`category` (`category.Category`) – The category, whose attractivities are of interest.

Returns
The list of attractivity attributes of the given category

Return type
list[*attractivity_attribute.AttractivityAttribute*]

`get_aggregation_methods`()

Returns a list of all aggregation methods that are available. This function returns all available aggregation methods, not just the ones that are active in the current project.

Returns
The list of the available aggregation methods

Return type
list[*aggregation_method_enum.AggregationMethod*]

`is_aggregation_method_active`(*method*)

Checks, whether an aggregation method is active in the currently selected project.

Parameters
`method` (`aggregation_method_enum.AggregationMethod`) – The aggregation method that is checked for.

Returns
True, if there is currently a project selected and the given aggregation method is active in it; False otherwise.

Return type
bool

`set_aggregation_method_active`(*method*, *active*)

Activates or deactivates an aggregation method (of the currently selected project). Activates the given method, if active=True and deactivates it otherwise.

Parameters

- `method` (`aggregation_method_enum.AggregationMethod`) – The aggregation method we want to deactivate/activate
- `active` (*`bool`*) – True, if we want to activate the given method; False, if we want to deactivate it.

Returns
True, if a project is currently selected and the aggregation method was (de-)activated successfully; False, otherwise.

Return type
bool

`start_calculations`(*starting_phase*)

Starts the calculations in the given calculation phase in the currently selected project. The calculation process is split in different calculation phases. This function starts the calculation in a given phase.

Parameters
`starting_phase` (`calculation_phase_enum.CalculationPhase`) – The phase, in which the calculation should start

Returns
The status of the calculation: RUNNING, if the calculation was started successfully. For details on the meaning of this return value, see CalculationState

Return type
calculation_state_enum.CalculationState

`get_calculation_state()`

Gives the current calculation state of the selected project.

Returns
Returns the current state of the calculation. For details see documentation of CalculationState.

Return type
calculation_state_enum.CalculationState

`get_current_calculation_phase()`

Returns the calculation phase of the currently selected project.

Returns
The phase, that is currently running. NONE, if no phase is currently running.

Return type
calculation_phase_enum.CalculationPhase

`get_current_calculation_process()`

Returns an approximation of the progress of the calculations in the currently selected project. The progress is given as a number between 0 and 1, where 0 indicates that the calculation has not started yet and 1 indicates, that the calculations are done.

Returns
The value of the approximation.

Return type
float

`cancel_calculations()`

Cancels the calculations of the currently selected project. The calculation phase that is currently running will be stopped.

Returns
True, if the calculation was canceled successfully; False, otherwise.

Return type
bool

`export_project(`*path*`)`

Exports the currently selected project. Before it will be exported, the project will be saved. The folders and files of the project are copied to the given destination.

Parameters
path (*pathlib.Path*) – The place in storage, where the project should be exported to.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage or there was no project selected.

Return type
bool

`export_calculations`(*path*)

Exports the result of the calculations of the currently selected project. The folders and files regarding the results of the calculations are copied to the given destination.

Parameters
`path` (*pathlib.Path*) – The place in storage, where the results should be exported to.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage, the calculations have not produced results yet or there was no project selected.

Return type
bool

`export_configurations`(*path*)

Exports the category file of the currently selected project. A list of categories in the current project is stored at the given destination.

Parameters
`path` (*pathlib.Path*) – The place in storage, where the categories should be stored at.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage or there was no project selected.

Return type
bool

`get_project_name`()

Gets the name of the currently selected project.

Returns
The name of the project

Return type
str

`set_project_name`(*name*)

Sets the name of the currently selected project

Parameters
`name` (*str*) – The new name of the project, may not contain line breaks.

Returns
True, if the name was changed successfully; False, if an error occurred: The name is not valid or no project was selected.

Return type
bool

`get_project_description`()

Gets the description of the currently selected project.

Returns
The description of the project

Return type
str

`set_project_description`(*description*)

Sets the description of the currently selected project.

Parameters
`description` (*str*) – The new description of the project, may contain line breaks.

Returns
True, if the description was changed successfully; False, otherwise.

Return type
bool

`get_project_default_folder`()

Gets the project default folder. The project default folder is the folder, where projects are stored by default.

Returns
The path to the project default folder

Return type
pathlib.Path

`set_project_default_folder`(*default_folder*)

Sets the project default folder. The project default folder is the folder, where projects are stored by default. Projects of an old default folder will not be copied over.

Parameters
`default_folder` (*pathlib.Path*) – The path to the new project default folder

Returns
True, if the default folder was set successfully; False if an error occurred: The path is not valid or occupied.

Return type
bool

`generate_cut_out_map`()

Generates a map of the data of the currently selected project. Using the cut-out file of the project, this function creates a map as a html-file of the project. The path to the html-file is returned.

Returns
The path to the file, where the map is stored.

Return type
pathlib.Path

`get_calculation_visualization`()

Generates a graphic that visualizes the results of the calculations of the currently selected project.

Returns
The resulting visualization as axes of the matplotlib library.

Return type
matplotlib.axes.Axes

src.osm_configurator.control.control_interface module

`class IControl`

Bases: `ABC`

This class provides a consistent interface for access to the control-package. It is a facade, to make access easy. The control manages the access to the module. That's why this interface should give access to all the features provided by the model.

`abstract get_list_of_passive_projects()`

Returns the list of (passive) projects, which are in the default project folder of the application.

Returns
The list of passive projects in the default project folder.

Return type
list[*passive_project.PassiveProject*]

`abstract load_project(`*path*`)`

Loads a project All relevant data of a project are verified and loaded in memory. All coming project-referring calls will be directed to the given project.

Parameters
path (*pathlib.Path*) – The path to the project folder of the project, to be loaded.

Returns
True, if the project was loaded successfully; False if an error occurred, while trying to load the project. An error happens, if the path is not pointing to a valid project folder or if the project has corrupted files.

Return type
bool

`abstract create_project(`*name*, *description*, *destination*`)`

Creates a new project with the given attributes and loads it. The model creates a new project folder at the given destination, all relevant files are generated and the project is loaded into memory.

Parameters

- **name** (*str*) – The name of the to-be-created project, may not contain any line-breaks.
- **description** (*str*) – The description of the to-be-created project. May contain line-breaks.
- **destination** (*pathlib.Path*) – The path to the location, where the project-folder of the project should be created.

Returns
True, if the project was created successfully; False if an error occurred. An error occurs, if the name of the project is not valid, if the destination-path is not valid or if the destination-location is already occupied.

Return type
bool

`abstract delete_passive_project(`*project*`)`

Deletes a project out of the default project folder.

Parameters
project (`passive_project.PassiveProject`) – The project, that is going to be deleted.

Returns
True, if the (passive) project has been deleted successfully; False otherwise: The project does not exist or the application has not the right permissions to delete the project.

Return type
bool

`abstract save_project()`

Saves the project. The currently selected project is stored on the disk. All progress made since the last saving are saved.

Returns
True, if the project was saved successfully; False if an error occurred, while attempting to save the project or when there is no project selected.

Return type
bool

`abstract set_current_config_phase(`*config_phase*`)`

Stores the current configuration phase in the model.

Parameters
config_phase (`config_phase_enum.ConfigPhase`) – The new configuration phase.

Returns
True, if setting the configuration phase was successful; False, otherwise.

Return type
bool

`abstract get_current_config_phase()`

Returns the configuration phase, that is currently stored in the model.

Returns
The configuration phase, that is currently stored in the model.

Return type
config_phase_enum.ConfigPhase

`abstract is_project_loaded()`

Checks, whether any project is currently loaded/selected.

Returns
True, if a project is currently selected; False, otherwise.

Return type
bool

`abstract set_osm_data_reference(`*path*`)`

Sets the reference to the osm-data for the selected project. The reference contains the osm-data used in the calculations of the project. This method does not check if the given data is valid.

Parameters
path (`pathlib.Path`) – The reference to the osm-data

Returns
True, if the new reference was set successfully; False, if an error occurred while setting the reference.

Return type
bool

`abstract get_osm_data_reference()`

Returns the path to the osm-data, that is used in the currently selected project.

Returns
The path to the osm-data of the currently selected project.

Return type
pathlib.Path

`abstract download_osm_data(`*path*`)`

Downloads osm-data The osm-data to be downloaded are defined by a geojson-file. The data is downloaded and the reference to the correct osm-files is stored.

Parameters
`path` (*`pathlib.Path`*) – The path to the geojson-file.

Returns
True on success, False otherwise

Return type
bool

`abstract get_cut_out_mode()`

Gets the method of how the geofilter shall cut out on the OSM-Data.

Returns
The cut-out-mode of the currently selected project.

Return type
cut_out_mode_enum.CutOutMode

`abstract set_cut_out_mode(`*mode*`)`

Sets the method of how the geofilter shall cut out on the OSM-Data.

Parameters
`mode` (`cut_out_mode_enum.CutOutMode`) – The mode, to be set

Returns
True, if the CutOutMode was set successfully; False, if an error occurred or no project is currently selected.

Return type
bool

`abstract set_cut_out_reference(`*path*`)`

Sets the reference to the cut-out file of the currently selected project. This file is later used to calculate the geofilter.

Parameters
`path` (*`pathlib.Path`*) – The path to the file containing the cut-out-geometries

Returns
True, if the reference was set successfully; False, if an error occurred. An error occurs, if no project is currently selected or if the given path is not valid or occupied.

Return type
bool

`abstract get_cut_out_reference()`

Gets the reference to the cut-out file of the currently selected project.

Returns
The current reference to the cut-out file.

Return type
pathlib.Path

`abstract check_conflicts_in_category_configuration`(*path*)

Checks for a given file, if it is a valid category-file and checks, whether there are naming conflicts with the categories of the currently selected project.

Parameters
path (*pathlib.Path*) – The path to the category-file

Returns
True, if there is currently a project selected and there are no naming conflicts; False, otherwise.

Return type
bool

`abstract import_category_configuration`(*path*)

Imports the given categories into the currently selected project. Adds the given categories to the category list of the project.

Parameters
path (*pathlib.Path*) – The path to the category file

Returns
True, if the categories where added successfully; False, if there is no project loaded, the category file is corrupted or the category file does not exist.

Return type
bool

`abstract get_list_of_categories`()

Returns the list of all categories, that are currently in the currently selected project.

Returns
A list of the categories of the project in no particular order.

Return type
list[*category.Category*]

`abstract create_category`()

Creates a new category in the currently selected project. A new category is added to the list of categories of the project. The category has empty properties, except for an arbitrary name. If the creation fails, none will be returned and there won't be a category added.

Returns
The newly created category, none if there was an error

Return type
category.Category

`abstract delete_category`(*category*)

Deletes the given category. Removes the given category from the list of categories of the currently selected project

Parameters
category (`category.Category`) – The category, to be deleted

Returns
True, if the category was deleted successfully; False, otherwise

Return type
bool

`abstract get_list_of_key_recommendations`(*current_input*)

Returns a list of recommended keys, based on the input that is already entered by the user.

Parameters
current_input (*str*) – The input, that is currently written by the user.

Returns
A list of key recommendations, based on the current_input.

Return type
list[str]

`abstract get_attractivities_of_category`(*category*)

Returns the attractivity attributes that are defined for the given category.

Parameters
category (`category.Category`) – The category, whose attractivities are of interest.

Returns
The list of attractivity attributes of the given category

Return type
list[*attractivity_attribute.AttractivityAttribute*]

`abstract get_aggregation_methods`()

Returns a list of all aggregation methods that are available. This function returns all available aggregation methods, not just the ones that are active in the current project.

Returns
The list of the available aggregation methods

Return type
list[*aggregation_method_enum.AggregationMethod*]

`abstract is_aggregation_method_active`(*method*)

Checks, whether an aggregation method is active in the currently selected project.

Parameters
method (`aggregation_method_enum.AggregationMethod`) – The aggregation method that is checked for.

Returns
True, if there is currently a project selected and the given aggregation method is active in it; False otherwise.

Return type
bool

`abstract set_aggregation_method_active`(*method*, *active*)

Activates or deactivates an aggregation method (of the currently selected project). Activates the given method, if active=True and deactivates it otherwise.

Parameters

- **method** (`aggregation_method_enum.AggregationMethod`) – The aggregation method we want to deactivate/activate
- **active** (*bool*) – True, if we want to activate the given method; False, if we want to deactivate it.

Returns
True, if a project is currently selected and the aggregation method was (de-)activated successfully; False, otherwise.

Return type
bool

`abstract start_calculations(`*starting_phase*`)`

Starts the calculations in the given calculation phase in the currently selected project. The calculation process is split in different calculation phases. This function starts the calculation in a given phase.

Parameters
`starting_phase` (`calculation_phase_enum.CalculationPhase`) – The phase, in which the calculation should start

Returns
The status of the calculation: RUNNING, if the calculation was started successfully. For details on the meaning of this return value, see CalculationState

Return type
calculation_state_enum.CalculationState

`abstract get_calculation_state()`

Gives the current calculation state of the selected project.

Returns
Returns the current state of the calculation. For details see documentation of CalculationState.

Return type
calculation_state_enum.CalculationState

`abstract get_current_calculation_phase()`

Returns the calculation phase of the currently selected project.

Returns
The phase, that is currently running. NONE, if no phase is currently running.

Return type
calculation_phase_enum.CalculationPhase

`abstract get_current_calculation_process()`

Returns an approximation of the progress of the calculations in the currently selected project. The progress is given as a number between 0 and 1, where 0 indicates that the calculation has not started yet and 1 indicates, that the calculations are done.

Returns
The value of the approximation.

Return type
float

`abstract cancel_calculations()`

Cancels the calculations of the currently selected project. The calculation phase that is currently running will be stopped.

Returns
True, if the calculation was canceled successfully; False, otherwise.

Return type
bool

`abstract export_project(path)`

Exports the currently selected project. Before it will be exported, the project will be saved. The folders and files of the project are copied to the given destination.

Parameters
`path` (`pathlib.Path`) – The place in storage, where the project should be exported to.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage or there was no project selected.

Return type
bool

`abstract export_calculations(path)`

Exports the result of the calculations of the currently selected project. The folders and files regarding the results of the calculations are copied to the given destination.

Parameters
`path` (`pathlib.Path`) – The place in storage, where the results should be exported to.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage, the calculations have not produced results yet or there was no project selected.

Return type
bool

`abstract export_configurations(path)`

Exports the category file of the currently selected project. A list of categories in the current project is stored at the given destination.

Parameters
`path` (`pathlib.Path`) – The place in storage, where the categories should be stored at.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage or there was no project selected.

Return type
bool

`abstract export_cut_out_map(path)`

Exports the map generated by the cut-out configuration.

Parameters
`path` (`pathlib.Path`) – The place in storage, where the cut-out-map should be stored at.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage, the application wasn't able to create the map or there was no project selected.

Return type
bool

`abstract get_project_name()`

Gets the name of the currently selected project.

Returns
The name of the project

Return type
str

`abstract set_project_name(`*name*`)`

Sets the name of the currently selected project

Parameters
`name` (*str*) – The new name of the project, may not contain line breaks.

Returns
True, if the name was changed successfully; False, if an error occurred: The name is not valid or no project was selected.

Return type
bool

`abstract get_project_description()`

Gets the description of the currently selected project.

Returns
The description of the project

Return type
str

`abstract set_project_description(`*description*`)`

Sets the description of the currently selected project.

Parameters
`description` (*str*) – The new description of the project, may contain line breaks.

Returns
True, if the description was changed successfully; False, otherwise.

Return type
bool

`abstract get_project_default_folder()`

Gets the project default folder. The project default folder is the folder, where projects are stored by default.

Returns
The path to the project default folder

Return type
pathlib.Path

`abstract set_project_default_folder(`*default_folder*`)`

Sets the project default folder. The project default folder is the folder, where projects are stored by default. Projects of an old default folder will not be copied over.

Parameters
`default_folder` (*pathlib.Path*) – The path to the new project default folder

Returns
True, if the default folder was set successfully; False if an error occurred: The path is not valid or occupied.

Return type
bool

`abstract generate_cut_out_map()`

Generates a map of the data of the currently selected project. Using the cut-out file of the project, this function creates a map as a html-file of the project. The path to the html-file is returned.

Returns
The path to the file, where the map is stored.

Return type
pathlib.Path

`abstract get_calculation_visualization()`

Generates a graphic that visualizes the results of the calculations of the currently selected project.

Returns
The resulting visualization as axes of the matplotlib library.

Return type
matplotlib.axes.Axes

src.osm_configurator.control.cut_out_controller module

`class CutOutController(`*model*`)`

Bases: `object`

The CutOutController is responsible for consistently forwarding requests to the model, concerning the cut-out filter of the currently selected project.

`__init__(`*model*`)`

Creates a new instance of the CutOutController, with an association to the model.

Parameters
`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

`get_cut_out_mode()`

Gets the method of how the geofilter shall cut out on the OSM-Data.

Returns
The cut-out-mode of the currently selected project.

Return type
cut_out_mode_enum.CutOutMode

`set_cut_out_mode(`*mode*`)`

Sets the method of how the geofilter shall cut out on the OSM-Data.

Parameters
`mode` (`cut_out_mode_enum.CutOutMode`) – The mode, to be set

Returns
True, if the CutOutMode was set successfully; False, if an error occurred or no project is currently selected.

Return type
bool

`set_cut_out_reference(`*path*`)`

Sets the reference to the cut-out file of the currently selected project. This file is later used to calculate the geofilter.

Parameters
`path` (*`pathlib.Path`*) – The path to the file containing the cut-out-geometries

Returns
True, if the reference was set successfully; False, if an error occurred. An error occurs, if no project is currently selected or if the given path is not valid or occupied.

Return type
bool

`get_cut_out_reference()`

” Gets the reference to the cut-out file of the currently selected project.

Returns
The current reference to the cut-out file.

Return type
pathlib.Path

src.osm_configurator.control.data_visualization_controller module

`class DataVisualizationController(`*model*`)`

Bases: `object`

The DataVisualizationController is responsible for forwarding requests to the model, regarding the visualization of data from the model.

`__init__(`*model*`)`

Creates a new instance of the DataVisualizationController, with an association to the model.

Parameters
`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

`generate_cut_out_map()`

Generates a map of the data of the currently selected project. Using the cut-out file of the project, this function creates a map as a html-file of the project. The path to the html-file is returned.

Returns
The path to the file, where the map is stored.

Return type
pathlib.Path

`get_calculation_visualization()`

Generates a graphic that visualizes the results of the calculations of the currently selected project.

Returns
The resulting visualization as axes of the matplotlib library.

Return type
matplotlib.axes.Axes

src.osm_configurator.control.export_controller module

`class ExportController`(*model*)

Bases: `object`

The ExportController forwards requests to the model, regarding the export of information as files, in the currently selected project.

`__init__`(*model*)

Creates a new instance of the ExportController, with an association to the model.

Parameters
`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

`export_project`(*path*)

Exports the currently selected project. Before it will be exported, the project will be saved. The folders and files of the project are copied to the given destination.

Parameters
`path` (*`pathlib.Path`*) – The place in storage, where the project should be exported to.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage or there was no project selected.

Return type
bool

`export_calculations`(*path*)

Exports the result of the calculations of the currently selected project. The folders and files regarding the results of the calculations are copied to the given destination.

Parameters
`path` (*`pathlib.Path`*) – The place in storage, where the results should be exported to.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage, the calculations have not produced results yet or there was no project selected.

Return type
bool

`export_configurations`(*path*)

Exports the category file of the currently selected project. A list of categories in the current project is stored at the given destination.

Parameters
`path` (*`pathlib.Path`*) – The place in storage, where the categories should be stored at.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage or there was no project selected.

Return type
bool

`export_cut_out_map`(*path*)

Exports the map generated by the cut-out configuration.

Parameters
`path` (*pathlib.Path*) – The place in storage, where the cut-out-map should be stored at.

Returns
True, if the export was successful; False, if an error occurred: The path was not valid or occupied, there was not enought space in storage, the application wasn't able to create the map or there was no project selected.

Return type
bool

src.osm_configurator.control.osm_data_controller module

class OSMDataController(*model*)

Bases: `object`

The OSMDataController is responsible for consistently forwarding requests regarding the OSM-data of the currently selected project.

__init__(*model*)

Creates a new instance of the OSMDataController, with an association to the model.

Parameters
`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

set_osm_data_reference(*path*)

Sets the reference to the osm-data for the selected project. The reference contains the osm-data used in the calculations of the project. This method does not check if the given data is valid.

Parameters
`path` (*pathlib.Path*) – The reference to the osm-data

Returns
True, if the new reference was set successfully; False, if an error occurred while setting the reference.

Return type
bool

get_osm_data_reference()

Returns the path to the osm-data, that is used in the currently selected project.

Returns
The path to the osm-data of the currently selected project.

Return type
pathlib.Path

download_osm_data(*path*)

Downloads osm-data The osm-data to be downloaded are defined by a geojson-file. The data is downloaded and the reference to the correct osm-files is stored.

Parameters
`path` (*pathlib.Path*) – The path to the geojson-file.

Returns
True on success, False otherwise

Return type
bool

src.osm_configurator.control.project_controller module

`class ProjectController`(*model*)

Bases: `object`

The ProjectController is responsible for consistently forwarding requests regarding the project management to the model. It is responsible for managing, saving, loading, deleting and creating projects.

`__init__`(*model*)

Creates a new instance of the ProjectController, with an association to the model.

Parameters
`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

`get_list_of_passive_projects`()

Returns the list of (passive) projects, which are in the default project folder of the application.

Returns
The list of passive projects in the default project folder.

Return type
list[*passive_project.PassiveProject*]

`load_project`(*path*)

Loads a project All relevant data of a project are verified and loaded in memory. All coming project-referring calls will be directed to the given project.

Parameters
`path` (*`pathlib.Path`*) – The path to the project folder of the project, to be loaded.

Returns
True, if the project was loaded successfully; False if an error occurred, while trying to load the project. An error happens, if the path is not pointing to a valid project folder or if the project has corrupted files.

Return type
bool

`create_project`(*name*, *description*, *destination*)

Creates a new project with the given attributes and loads it. The model creates a new project folder at the given destination, all relevant files are generated and the project is loaded into memory.

Parameters

- `name` (*`str`*) – The name of the to-be-created project, may not contain any line-breaks.
- `description` (*`str`*) – The description of the to-be-created project. May contain line-breaks.
- `destination` (*`pathlib.Path`*) – The path to the location, where the project-folder of the project should be created.

Returns
True, if the project was created successfully; False if an error occurred. An error occurs, if the name of the project is not valid, if the destination-path is not valid or if the destination-location is already occupied.

Return type
bool

`delete_passive_project`(*project*)

Deletes a project out of the default project folder.

Parameters
`project` (`passive_project.PassiveProject`) – The project, that is going to be deleted.

Returns
True, if the (passive) project has been deleted successfully; False otherwise: The project does not exist or the application has not the right permissions to delete the project.

Return type
bool

`save_project`()

Saves the project. The currently selected project is stored on the disk. All progress made since the last saving are saved.

Returns
True, if the project was saved successfully; False if an error occurred, while attempting to save the project or when there is no project selected.

Return type
bool

`set_current_config_phase`(*config_phase*)

Stores the current configuration phase in the model.

Parameters
`config_phase` (`config_phase_enum.ConfigPhase`) – The new configuration phase.

Returns
True, if setting the configuration phase was successful; False, otherwise.

Return type
bool

`get_current_config_phase`()

Returns the configuration phase, that is currently stored in the model.

Returns
The configuration phase, that is currently stored in the model.

Return type
config_phase_enum.ConfigPhase

`is_project_loaded`()

Checks, whether any project is currently loaded/selected.

Returns
True, if a project is currently selected; False, otherwise.

Return type
bool

src.osm_configurator.control.settings_controller module

`class SettingsController`(*model*)

Bases: `object`

The SettingsController is responsible for forwarding requests to the model, regarding the settings of the application and the currently selected project.

`__init__`(*model*)

Creates a new instance of the SettingsController, with an association to the model.

Parameters
`model` (`application_interface.IApplication`) – The interface which is used to communicate with the model.

`get_project_name()`

Gets the name of the currently selected project.

Returns
The name of the project

Return type
str

`set_project_name`(*name*)

Sets the name of the currently selected project

Parameters
`name` (*str*) – The new name of the project, may not contain line breaks.

Returns
True, if the name was changed successfully; False, if an error occurred: The name is not valid or no project was selected.

Return type
bool

`get_project_description()`

Gets the description of the currently selected project.

Returns
The description of the project

Return type
str

`set_project_description`(*description*)

Sets the description of the currently selected project.

Parameters
`description` (*str*) – The new description of the project, may contain line breaks.

Returns
True, if the description was changed successfully; False, otherwise.

Return type
bool

`get_project_default_folder()`

Gets the project default folder. The project default folder is the folder, where projects are stored by default.

Returns
The path to the project default folder

Return type
pathlib.Path

`set_project_default_folder`(*default_folder*)

Sets the project default folder. The project default folder is the folder, where projects are stored by default. Projects of an old default folder will not be copied over.

Parameters
`default_folder` (*`pathlib.Path`*) – The path to the new project default folder

Returns
True, if the default folder was set successfully; False if an error occurred: The path is not valid or occupied.

Return type
bool

Module contents

src.osm_configurator.model package

Subpackages

src.osm_configurator.model.application package

Submodules

src.osm_configurator.model.application.application module

`class Application`

Bases: *`IApplication`*

The IApplication job, is to provide the functionality the application needs.

`__init__`()

Creates a new instance of the application_interface.Application.

`get_passive_project_list`()

Returns the list of all passive project in the current project default folder.

Returns
The list of the passive projects.

Return type
list[*passive_project.PassiveProject*]

`get_key_recommendation`(*input*)

Creates recommendations based on user input

Parameters
`input` (*`str`*) – The input from which to generate suggestions.

Returns
Returns a list of strings containing the recommendations depending on the input.

Return type
list[str]

`create_project`(*name*, *description*, *destination*)

This method creates a new project with a name, a description and saves it at a given destination.

Parameters

- **name** (*str*) – The name of the new project.
- **description** (*str*) – The description of the new project.
- **destination** (*pathlib.Path*) – The path, where the new project should be saved.

Returns
True if create_project completed successfully, otherwise false.

Return type
bool

`load_project`(*path*)

This method loads an existing project. This project can be internal or external ones. The path is pointing towards the folder, where the project is saved.

Parameters
path (*pathlib.Path*) – The path of the project, to be loaded.

Returns
True if loading the project is working, otherwise false.

Return type
bool

`start_calculation`(*calculation_phase*)

This method is to start the calculation (after the configuration is finished).

Parameters
calculation_phase (`calculation_phase_enum.CalculationPhase`) – The calculation phase, where the calculation shall start.

Returns
The calculation state where the calculation started. Can be an error state, so signify an error, that prevents the start of the calculation.

Return type
CalculationState

`get_osm_data`()

Gives back the path pointing towards the OSM data file.

Returns
The path pointing towards the OSM data.

Return type
pathlib.Path

`set_osm_data`(*osm_data*)

Edits the path pointing towards the OSM data file.

Parameters
osm_data (*pathlib.Path*) – The new path towards the osm data file.

Returns
True if changing the path works, otherwise false.

Return type
bool

`get_all_aggregation_methods()`

Gives back a List of all possible aggregation methods.

Returns
A list containing all aggregation methods.

Return type
list[*aggregation_method_enum.AggregationMethod*]

`is_aggregation_method_active(`*method*`)`

Checks, if a given aggregation method is active.

Parameters
`method` (`aggregation_method_enum.AggregationMethod`) – The method, which is to be checked.

Returns
True if the aggregation method is active, otherwise false.

Return type
bool

`set_aggregation_method_active(`*method*, *active*`)`

Changes the aggregation method from active to inactive and vice versa. If an already active aggregation method should be activated, it stays active. The same applies to inactive aggregation methods, which should be deactivated.

Parameters

- **`method`** (`aggregation_method_enum.AggregationMethod`) – The method, which state should be changed.
- **`active`** (*`bool`*) – This is the new state of the aggregation method.

Returns
True if changing the state works, otherwise false.

Return type
bool

`get_cut_out_mode()`

Gives back the used cut-out mode.

Returns
The used cut-out mode.

Return type
cut_out_mode_enum.CutOutMode

`set_cut_out_mode(`*new_cut_out_mode*`)`

Changes the cut-out mode used during the reduction phase in the calculation.

Parameters
`new_cut_out_mode` (`cut_out_mode_enum.CutOutMode`) – The new cut-out mode for the calculation.

Returns
True if changing the cut-out mode works, otherwise false.

Return type
bool

`get_cut_out_path()`

Gives back the path pointing towards the cut-out file.

Returns
The path pointing towards the cut-out.

Return type
pathlib.Path

`set_cut_out_path`(*path*)

Changes the path pointing towards the cut-out file.

Parameters
`path` (*pathlib.Path*) – The new path, towards a cut-out file.

Returns
True if changing the cut-out path works, otherwise false.

Return type
bool

`get_category`(*index*)

Gets a category based on the index.

Parameters
`index` (*int*) – Index in the categories-list, that will be returned.

Returns
The Category we wanted, NONE if the index is out of bounds of the list.

Return type
category.Category

`get_categories()`

Getter for all the Categories.

Returns
List of the chosen categories.

Return type
list[*category.Category*]

`create_category()`

Creates a new category, that will be empty.

Returns
The newly created category.

Return type
category.Category

`remove_category`(*category*)

Removes the given category from the categories list, if element is inside the List.

Parameters
`category` (`category.Category`) – Category that will be removed.

Returns
True, if the element was removed correctly, else false.

Return type
bool

`override_categories`(*new_category_list*)

Overwrites the list of categories with the given list, if both lists are not identical.

Parameters
`new_category_list` (`list[category.Category]`) – List of categories, that will overwrite the already existing list.

Returns
True, if the replacement was successful, else false.

Return type
bool

`merge_categories`(*category_input_list*)

Merges the existing category list with the given list if both lists are not identical. If two categories conflict in their name, the newer category will be used.

Parameters
`category_input_list` (`list[category.Category]`) – New list of categories that will be merged into the existing list.

Returns
True, if the merging was successful, else False.

Return type
bool

`create_map`(*cut_out*)

This method to create a map from to given cut-out.

Parameters
`cut_out` (`cut_out_configuration.CutOutConfiguration`) – The cut-out configuration from which the map should be created.

Returns
True if creating the map works, otherwise false.

Return type
bool

`create_boxplot`(*data*)

This method is to visualize the data by creating a boxplot. It is used to visualize the calculated end result via a boxplot.

Parameters
`data` (`matplotlib.axes.Axes`) – A plot of the data which we want to visualize.

Returns
True if creating the boxplot works, otherwise false.

Return type
bool

`get_location`()

Getter for the location of the active project on the disk.

Returns
The location of the active project

Return type
pathlib.Path

`set_name`(*new_name*)

This method changes the name of the project.

Parameters
`new_name` (*str*) – The new name of the project

Returns
true if change was successful, false else

Return type
bool

`get_name`()

This method returns the name of the project.

Returns
name of the project

Return type
str

`set_description`(*new_description*)

This method changes the description of the project.

Parameters
`new_description` (*str*) – The new description of the project

Returns
true if change successful, false else

Return type
bool

`get_description`()

This method returns the description of the project.

Returns
The description of the project

Return type
str

`export_project`(*path*)

Exports the whole project to the given path.

Parameters
`path` (*pathlib.Path*) – The path where the project shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

`export_configuration`(*path*)

Exports the configuration to the given path. More specific, exports all the categories and their configurations, to the given path.

Parameters
`path` (*pathlib.Path*) – The path where the configurations shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

`export_calculation`(*path*)

Exports the results of the calculation to the given path. Whereby the calculation are a folder with all the different results from each calculation step in it.

Parameters
`path` (*pathlib.Path*) – The path where the results of the calculation shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

`export_map`(*path*)

Exports an HTML-Data with the map in it, to the given path.

Parameters
`path` (*pathlib.Path*) – The path, where the map shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

src.osm_configurator.model.application.application_interface module

`class IApplication`

Bases: ABC

The IApplication job, is to provide the functionality the application needs.

`abstract get_passive_project_list`()

Returns the list of all passive project in the current project default folder.

Returns
The list of the passive projects.

Return type
list[*passive_project.PassiveProject*]

`abstract get_key_recommendation`(*input*)

Creates recommendations based on user input

Parameters
`input` (*str*) – The input from which to generate suggestions.

Returns
Returns a list of strings containing the recommendations depending on the input.

Return type
list[str]

`abstract create_project`(*name*, *description*, *destination*)

This method creates a new project with a name, a description and saves it at a given destination.

Parameters

- **`name`** (*`str`*) – The name of the new project.
- **`description`** (*`str`*) – The description of the new project.
- **`destination`** (*`pathlib.Path`*) – The path, where the new project should be saved.

Returns

True if create_project completed successfully, otherwise false.

Return type

bool

`abstract load_project`(*path*)

This method loads an existing project. This project can be internal or external ones. The path is pointing towards the folder, where the project is saved.

Parameters

`path` (*`pathlib.Path`*) – The path of the project, to be loaded.

Returns

True if loading the project is working, otherwise false.

Return type

bool

`abstract start_calculation`(*calculation_phase*)

This method is to start the calculation (after the configuration is finished).

Parameters

`calculation_phase` (`calculation_phase_enum.CalculationPhase`) – The calculation phase, where the calculation shall start.

Returns

The calculation state where the calculation started. Can be an error state, so signify an error, that prevents the start of the calculation.

Return type

CalculationState

`abstract get_osm_data`()

Gives back the path pointing towards the OSM data file.

Returns

The path pointing towards the OSM data.

Return type

pathlib.Path

`abstract set_osm_data`(*osm_data*)

Edits the path pointing towards the OSM data file.

Parameters

`osm_data` (*`pathlib.Path`*) – The new path towards the osm data file.

Returns

True if changing the path works, otherwise false.

Return type
bool

`abstract get_all_aggregation_methods()`

Gives back a List of all possible aggregation methods.

Returns
A list containing all aggregation methods.

Return type
list[*aggregation_method_enum.AggregationMethod*]

`abstract is_aggregation_method_active`(*method*)

Checks, if a given aggregation method is active.

Parameters
`method` (`aggregation_method_enum.AggregationMethod`) – The method, which is to be checked.

Returns
True if the aggregation method is active, otherwise false.

Return type
bool

`abstract set_aggregation_method_active`(*method*, *active*)

Changes the aggregation method from active to inactive and vice versa. If an already active aggregation method should be activated, it stays active. The same applies to inactive aggregation methods, which should be deactivated.

Parameters

- **`method`** (`aggregation_method_enum.AggregationMethod`) – The method, which state should be changed.
- **`active`** (*`bool`*) – This is the new state of the aggregation method.

Returns
True if changing the state works, otherwise false.

Return type
bool

`abstract get_cut_out_mode()`

Gives back the used cut-out mode.

Returns
The used cut-out mode.

Return type
cut_out_mode_enum.CutOutMode

`abstract set_cut_out_mode`(*new_cut_out_mode*)

Changes the cut-out mode used during the reduction phase in the calculation.

Parameters
`new_cut_out_mode` (`cut_out_mode_enum.CutOutMode`) – The new cut-out mode for the calculation.

Returns
True if changing the cut-out mode works, otherwise false.

Return type
bool

`abstract get_cut_out_path()`

Gives back the path pointing towards the cut-out file.

Returns
The path pointing towards the cut-out.

Return type
pathlib.Path

`abstract set_cut_out_path(`*path*`)`

Changes the path pointing towards the cut-out file.

Parameters
path (*pathlib.Path*) – The new path, towards a cut-out file.

Returns
True if changing the cut-out path works, otherwise false.

Return type
bool

`abstract get_category(`*index*`)`

Gets a category based on the index.

Parameters
index (*int*) – Index in the categories-list, that will be returned.

Returns
The Category we wanted, NONE if the index is out of bounds of the list.

Return type
category.Category

`abstract get_categories()`

Getter for all the Categories.

Returns
List of the chosen categories.

Return type
list[*category.Category*]

`abstract create_category()`

Creates a new category, that will be empty.

Returns
The newly created category.

Return type
category.Category

`abstract remove_category(`*category*`)`

Removes the given category from the categories list, if element is inside the List.

Parameters
category (`category.Category`) – Category that will be removed.

Returns
True, if the element was removed correctly, else false.

Return type
bool

`abstract override_categories`(*new_category_list*)

Overwrites the list of categories with the given list, if both lists are not identical.

Parameters
new_category_list (*list[category.Category]*) – List of categories, that will overwrite the already existing list.

Returns
True, if the replacement was successful, else false.

Return type
bool

`abstract merge_categories`(*category_input_list*)

Merges the existing category list with the given list if both lists are not identical. If two categories conflict in their name, the newer category will be used.

Parameters
category_input_list (*list[category.Category]*) – New list of categories that will be merged into the existing list.

Returns
True, if the merging was successful, else False.

Return type
bool

`abstract create_map`(*cut_out*)

This method to create a map from to given cut-out.

Parameters
cut_out (`cut_out_configuration.CutOutConfiguration`) – The cut-out configuration from which the map should be created.

Returns
True if creating the map works, otherwise false.

Return type
bool

`abstract create_boxplot`(*data*)

This method is to visualize the data by creating a boxplot. It is used to visualize the calculated end result via a boxplot.

Parameters
data (*matplotlib.axes.Axes*) – A plot of the data which we want to visualize.

Returns
True if creating the boxplot works, otherwise false.

Return type
bool

`abstract get_location`()

Getter for the location of the active project on the disk.

Returns
The location of the active project

Return type
pathlib.Path

`get_default_location()`

Gives back the path pointing towards the project.

Returns
Returns the path of the default location.

Return type
pathlib.Path

`set_default_location(`*new_location*`)`

Sets the default path pointing towards the project to a new Location.

Parameters
`new_location` (*`pathlib.Path`*) – The new Location, where the user wants to save new projects.

`abstract set_name(`*new_name*`)`

This method changes the name of the project.

Parameters
`new_name` (*`str`*) – The new name of the project

Returns
true if change was successful, false else

Return type
bool

`abstract get_name()`

This method returns the name of the project.

Returns
name of the project

Return type
str

`abstract set_description(`*new_description*`)`

This method changes the description of the project.

Parameters
`new_description` (*`str`*) – The new description of the project

Returns
true if change successful, false else

Return type
bool

`abstract get_description()`

This method returns the description of the project.

Returns
The description of the project

Return type
str

abstract export_project(*path*)

Exports the whole project to the given path.

Parameters
path (*pathlib.Path*) – The path where the project shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

abstract export_configuration(*path*)

Exports the configuration to the given path. More specific, exports all the categories and their configurations, to the given path.

Parameters
path (*pathlib.Path*) – The path where the configurations shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

abstract export_calculation(*path*)

Exports the results of the calculation to the given path. Whereby the calculation are a folder with all the different results from each calculation step in it.

Parameters
path (*pathlib.Path*) – The path where the results of the calculation shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

abstract export_map(*path*)

Exports an HTML-Data with the map in it, to the given path.

Parameters
path (*pathlib.Path*) – The path, where the map shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

src.osm_configurator.model.application.application_settings module

class ApplicationSettings

Bases: `object`

This class job is to manage the settings apart from the project settings. In those settings the default-location to save projects can be changed.

__init__()

Creates a new instance of the ApplicationSettings.

get_default_location()

Gives back the path pointing towards the project.

Returns
Returns the path of the default location.

Return type
pathlib.Path

set_default_location(*new_location*)

Sets the default path pointing towards the project to a new Location.

Parameters
new_location (*pathlib.Path*) – The new Location, where the user wants to save new projects.

src.osm_configurator.model.application.passive_project module

class PassiveProject(*project_folder_path*)

Bases: `object`

This class job is to manage the passive projects. Those are the all projects shown in the Main Menu. Therefore, the class holds the name, description, last edit date and path of the projects.

__init__(*project_folder_path*)

Creates a new instance of the PassiveProject.

Parameters
project_folder_path (*Path*) – The path to the project you want to make a PassiveProject on.

get_name()

Gives back the name of the passive project.

Returns
The name of the passive project.

Return type
str

get_description()

Gives back the description of the passive project.

Returns
The description of the passive project.

Return type
str

get_edit_date()

Gives back the last edit date of the passive project.

Returns
The last edit date of the passive project.

Return type
str

`get_project_folder_path()`

Gives back the path pointing towards the passive project.

Returns
The path pointing towards the passive project.

Return type
pathlib.Path

src.osm_configurator.model.application.recommender_system module

`class RecommenderSystem`

Bases: `object`

This class job is to provide recommendations to different classes.

`__init__()`

Creates a new instance of the RecommenderSystem.

`recommend`(*input*)

Creates recommendations based on user input

Parameters
`input` (*`str`*) – The input from which to generate suggestions.

Returns
Returns a list of strings containing the recommendations depending on the input.

Return type
list<str>

Module contents

src.osm_configurator.model.parser package

Submodules

src.osm_configurator.model.parser.calculation_parser module

`class CalculationParser`

Bases: *`CalculationParserInterface`*

The CalculationParser job is to parse the calculation files, that are created from the calculation process. It ensures that all data required for a calculation step is there.

Examples: The `TagFilterPhase` needs the files that got previously calculated in `GeoDataPhase`.

`__init__()`

Creates a new instance of the calculation_parser_interface.CalculationParser.

`check_validity_of_calculation_step`(*project_path*, *starting_point*)

Checks whether the passed starting_point is valid. A starting_point is valid when all calculation files required for the next step are present in the project.

Parameters

- **project_path** (*pathlib.Path*) – The path pointing towards the project, that needs to be validated.
- **starting_point** (calculation_phase_enum.CalculationPhase) – The step we want to calculate next.

Returns
True when starting_point is valid, otherwise false.

Return type
bool

src.osm_configurator.model.parser.calculation_parser_interface module

class CalculationParserInterface

Bases: ABC

The CalculationParser job is to parse the calculation files, that are created from the calculation process. It ensures that all data required for a calculation step is there.

Examples: The `TagFilterPhase` needs the files that got previously calculated in `GeoDataPhase`.

abstract check_validity_of_calculation_step(*project_path*, *starting_point*)

Checks whether the passed starting_point is valid. A starting_point is valid when all calculation files required for the next step are present in the project.

Parameters

- **project_path** (*pathlib.Path*) – The path pointing towards the project, that needs to be validated.
- **starting_point** (calculation_phase_enum.CalculationPhase) – The step we want to calculate next.

Returns
True when starting_point is valid, otherwise false.

Return type
bool

src.osm_configurator.model.parser.category_parser module

class CategoryParser

Bases: *CategoryParserInterface*

The CategoryParser job, is to parse the category file that are created when creating a project and make an internal representation out of it. In the category file there are the different categories from the project defined, for more information about this look at the documentation of `Category`.

__init__()

Creates a new instance of the CategoryParser.

parse_category_file(*path*)

Creates an internal representation of the category file it got as an input. What the Category includes, check this: `Category`.

Parameters
path (*pathlib.Path*) – The path to the category file.

Returns
A List of categories, that describe each category from the category file.

Return type
list[*category.Category*]

src.osm_configurator.model.parser.category_parser_interface module

class CategoryParserInterface

Bases: `ABC`

The CategoryParser job, is to parse the category file that are created when creating a project and make an internal representation out of it. In the category file there are the different categories from the project defined, for more information about this look at the documentation of `Category`.

abstract parse_category_file(*path*)

Creates an internal representation of the category file it got as an input. What the Category includes, check this: `Category`.

Parameters
path (*pathlib.Path*) – The path to the category file.

Returns
A List of categories, that describe each category from the category file.

Return type
list[*category.Category*]

src.osm_configurator.model.parser.cutOut_parser module

class CutOutParser

Bases: *CutOutParserInterface*

This Class parses cut_out files to an internal representation of the cut_out_file.

parse_cutout_file(*path*)

This method takes in the path to a cut_out file and parses to an internal representation of TrafficCells.

A cut_out file is a *.geojson* file that consists of multiple TrafficCells. Each TrafficCell has a name and a polygon, which is the bounding box of the Traffic Cell.

Parameters
path (*pathlib.Path*) – The path pointing towards cut_out file we want to parse.

Returns
Our cut_out file transformed into a list of TrafficCells.

Return type
list[*traffic_cell.TrafficCell*]

Examples

To see an example for a cut_out file check out the file *data/partOfKarlsruhe.geojson*.

src.osm_configurator.model.parser.cutOut_parser_interface module

`class CutOutParserInterface`

Bases: `ABC`

This Class parses cut_out files to an internal representation of the cut_out_file.

`abstract parse_cutout_file`(*path*)

This method takes in the path to a cut_out file and parses to an internal representation of TrafficCells.

A cut_out file is a *.geojson* file that consists of multiple TrafficCells. Each TrafficCell has a name and a polygon, which is the bounding box of the Traffic Cell.

Parameters

`path` (*`pathlib.Path`*) – The path pointing towards cut_out file we want to parse.

Returns

Our cut_out file transformed into a list of TrafficCells.

Return type

list[*traffic_cell.TrafficCell*]

Examples

To see an example for a cut_out file check out the file *data/partOfKarlsruhe.geojson*.

src.osm_configurator.model.parser.osm_data_parser module

`class CategoryParser`

Bases: *`OSMDataParserInterface`*

The OSMDataParser job is to parse the OSMData into a human-readable format. This human-readable format is a GeoDataFrame from GeoPandas.

`__init__`()

Creates a new instance of the CategoryParser.

`parse_osm_data_file`(*path*)

It gets a path pointing towards an OSM data in protocol buffer Binary Format(pbf) and transforms it into an GeoDataFrame. Each row in the GeoDataFrame is a single data entry, which is an osm element from the read osm data. Each column in the GeoDataFrame is a feature of the osm element from the osm_data, such as the location of the osm element, whereby a feature is a tag or something otherwise that describes the osm-element e.g. location.

Parameters

`path` (*`pathlib.Path`*) – The path pointing towards the OSM data we want to parse in the ".pbf" format.

Returns

The parsed OSM data as a GeoDataFrame.

Return type

GeoDataFrame

src.osm_configurator.model.parser.osm_data_parser_interface module

class OSMDataParserInterface

Bases: `ABC`

The OSMDataParser job is to parse the OSMData into a human-readable format. This human-readable format is a GeoDataFrame from GeoPandas.

abstract parse_osm_data_file(*path*)

It gets a path pointing towards an OSM data in protocol buffer Binary Format(pbf) and transforms it into an GeoDataFrame. Each row in the GeoDataFrame is a single data entry, which is an osm element from the read osm data. Each column in the GeoDataFrame is a feature of the osm element from the osm_data, such as the location of the osm element, whereby a feature is a tag or something otherwise that describes the osm-element e.g. location.

Parameters
path (*pathlib.Path*) – The path pointing towards the OSM data we want to parse in the ".pbf" format.

Returns
The parsed OSM data as a GeoDataFrame.

Return type
GeoDataFrame

Module contents

src.osm_configurator.model.project package

Subpackages

src.osm_configurator.model.project.calculation package

Submodules

src.osm_configurator.model.project.calculation.aggregation_method_enum module

class AggregationMethod(*value*)

Bases: `Enum`

This enum describes all the available aggregation methods that are possible to use. Whereby an aggregation methods is a method, that takes in data and an attractivity attribute. Finally, it outputs and calculates a function on these parameters. The first argument points towards the function, while the second argument is the name of the method.

SUM = (<function _sum>, 'sum')

Calculates the sum of the attractivity attribute over all osm elements from the data.

AVERAGE = (<function _average>, 'average')

Calculates the average of the attractivity attribute over all osm elements from the data.

MEAN = (<function _mean>, 'mean')

Calculates the mean of the attractivity attribute over all osm elements from the data.

`UPPER_QUARTILE = (<function _upper_quartile>, 'upper quartile')`

Calculates the upper_quartile of the attractivity attribute over all osm elements from the data.

`LOWER_QUARTILE = (<function _lower_quartile>, 'lower quartile')`

Calculates the lower quartile of the attractivity attribute over all osm elements from the data.

`MAXIMUM = (<function _maximum>, 'maximum')`

Calculates the maximum of the attractivity attribute over all osm elements from the data.

`MINIMUM = (<function _minimum>, 'minimum')`

Calculates the minimum of the attractivity attribute over all osm elements from the data.

`calculate_aggregation`(*data*, *attractivity_name*)

Executes the aggregation method of the called enum type.

Parameters

- `data` (*geopandas.GeoDataFrame*) – The data on which we want to execute the function on, should be a GeoDataFrame containing osm elements.
- `attractivity_name` (*str*) – This is the name of the attractivity through which we want to call the function, the attractivity_name should be the name of a column in the data GeoDataFrame.

Returns
The aggregated value of the attractivity values from all osm elements.

Return type
float

`get_name`()

Getter for the name of the enum type.

Returns
Name of the enum type

Return type
str

src.osm_configurator.model.project.calculation.aggregation_phase module

class AggregationPhase

Bases: *ICalculationPhase*

This calculation phase is responsible for aggregating the attractivity attributes in the given traffic cells. For details see the method calculate().

`calculate`(*configuration_manager*)

Aggregates the attractivity attributes in the given traffic cells. The calculation phase reads the data of the previous calculation phase. Now for every traffic cell all selected aggregation methods are performed for all attractivity attributes. For details on the different aggregation methods, see AggregationMethod. After the calculations are done, the results are stored on the hard-drive.

Parameters
`configuration_manager` (`configuration_manager.ConfigurationManager`) – The object containing all the configuration needed for execution.

Returns
The state of the calculation, after this phase finished its execution or failed trying so.

Return type

calculation_state_enum.CalculationState

src.osm_configurator.model.project.calculation.attractivity_phase module

class AttractivityPhase

Bases: *ICalculationPhase*

This calculation phase is responsible for calculating the attractivity attributes of the OSM-elements. For details see the method calculate().

calculate(*configuration_manager*)

Calculates the attractivity attributes of the osm-elements The calculation phase reads the data of the previous calculation phase. Now it calculates the attractivity attributes of every OSM-element. The attractivity attributes that are calculated for an osm-element are dependent on the category, the element belongs to. The value of an attractivity attribute is computed as a linear function with the previously computed attributes. The factors of this linear function are given in the configuration of the category. After the calculations are done, the results are stored on the hard-drive.

Parameters

configuration_manager (configuration_manager.ConfigurationManager) – The object containing all the configuration needed for execution.

Returns

The state of the calculation, after this phase finished its execution or failed trying so.

Return type

calculation_state_enum.CalculationState

src.osm_configurator.model.project.calculation.building_on_edge_manager module

class BuildingOnEdgeManager(*file_paths*, *border*)

Bases: object

This class handles the edge-case when buildings are on the edge of specified bounding-box. It is mainly used to remove building that are on this edge.

__init__(*file_paths*, *border*)

Creates a new instance of the "BuildingOnEdgeManager".

Parameters

- **file_paths** (*List[pathlib.Path]*) – A list of path each pointing towards an osm-data file through which we want to remove the buildings on the edge.
- **border** (*List[shapely.Polygon]*) – A list of polygon. Each polygon belongs to one entry in the file_path list and specifies the border of said file.

remove_buildings_on_edge()

Reads in the data from the file path it got, and removes all buildings from the data that are on the edge. This means building which are on the edge, half in half out.

Returns

True when successful, otherwise false.

Return type
(bool)

src.osm_configurator.model.project.calculation.calculation_manager module

`class CalculationManager(`*configuration_manager*`)`

Bases: `object`

The CalculationManager manages the calculation of the Project. The Calculation are distributed on the calculation phases which are also managed by the CalculationController.

`__init__(`*configuration_manager*`)`

Gets called when we first create an object of this class. It saves all information it needs for managing the calculations.

Parameters
configuration_manager (`configuration_manager.ConfigurationManager`) – Saves all information required to configure the calculation.

`cancel_calculation()`

This method will cancel an ongoing calculation. A calculation consists of an CalculationPhase, that will be interrupted.

Returns
True if it is successful and false if something goes wrong, or no calculation is going on.

Return type
bool

`start_calculation(`*starting_point*`)`

Starts the calculation. Distributes the calculations to the calculation phases.

Parameters
starting_point (`calculation_phase_enum.CalculationPhase`) – The starting phase of the calculation. The calculations start from this phase.

Returns
The state of the calculation, after trying to start the calculations.

Return type
calculation_phase_enum.CalculationState

src.osm_configurator.model.project.calculation.calculation_phase_enum module

`class CalculationPhase(`*value*`)`

Bases: `Enum`

This enum provides a list of phases the calculation can be in. These phases will be worked through in order like a pipeline. An enum consists of two values a name which will be displayed and an order in which the phases will be calculated, also used to know in which order to display the phases. If you want to know more about the calculation phases refer to the “Pflichtenheft”, or the documentation of the individual phases in *project.calculation*

Each enum consists of three variables: (name, folder_name, order). The name of the phase is used in the GUI, to display the correct name of the phase. The folder_name is used to save the results of each phase in it. The order is used to differentiate in which order the phases get called.

`NONE = ('None', 'none', 0)`

`GEO_DATA_PHASE = ('Data Input and Geo-Filter', 'geo_data_phase_results', 1)`

`TAG_FILTER_PHASE = ('Tag-filter', 'tag_filter_phase', 2)`

`REDUCTION_PHASE = ('Reduction', 'reduction_phase_results', 3)`

`ATTRACTIVITY_PHASE = ('Attractivity', 'attractivity_phase_results', 4)`

`AGGREGATION_PHASE = ('Aggregation', 'aggregation_phase_result', 5)`

`get_name()`

Getter for the name of the enum type.

Returns
Name of the Phase.

Return type
(str)

`get_folder_name_for_results()`

Getter for the folder name of the enum type.

Returns
The folder name of the enum.

Return type
(str)

`get_order()`

Getter for the order of the enum type.

Returns
order of the enum.

Return type
(int)

src.osm_configurator.model.project.calculation.calculation_phase_interface module

`class ICalculationPhase`

Bases: `ABC`

This class represents a calculation phase. A calculation phase is a single step in the big process of computing the final results. Calculation phases are executed after each other. A calculation phase consists of the following 3 steps:

1. Load needed results of previously computed calculation phases.
2. Execute the computations of this calculation phase.
3. Store the results of this computation phase so the following execution phases can read it.

`abstract calculate(`*configuration_manager*`)`

Performs the calculations of the calculation phase. This consists of the following steps:

1. Load needed results of previously computed calculation phases.
2. Execute the computations of this calculation phase.

3. Store the results of this computation phase so the following execution phases can read it.

   Parameters
   `configuration_manager` (`configuration_manager.ConfigurationManager`) – The ConfigurationManager where the information about the configuration of the configuration is stored.

   Returns
   The state of the calculation after this phase finished its execution or failed trying so.

   Return type
   calculation_state_enum.CalculationState

src.osm_configurator.model.project.calculation.calculation_state_enum module

`class CalculationState`(*value*)

Bases: `Enum`

This enum provides a list of states the calculations can be in. The states can be positive, indicating the calculation is working correctly. But they can be negative as well, indicating an error in the calculation. Every state is defined by a unique description of the state.

`NOT_STARTED_YET = ('Not started yet', 'The calculation was not started yet.')`

`RUNNING = ('Running', 'The calculations are currently running.')`

`ENDED_SUCCESSFULLY = ('Done', 'The calculations ended successfully.')`

`ERROR_INVALID_OSM_DATA = ('Invalid OSM Data', 'Error:  The osm data are not valid.')`

`ERROR_INVALID_CUT_OUT_DATA = ('Invalid Cut Out Data', 'Error:  The cut out data are not valid.')`

`ERROR_INVALID_CATEGORIES = ('Invalid Categories', 'Error:  The category configuration is not valid.')`

`ERROR_INVALID_PREVIOUS_CALCULATIONS = ('Invalid calculation phase', "Error:  This calculation phase can not be calculated, because a previous calculation has invalid results or wasn't run.")`

`get_name()`

Gives back the name of a calculation state.

Returns
The name of the calculation state.

Return type
str

`get_description()`

Gives back the description of a calculation state.

Returns
The description, that describes the state in natural language.

Return type
str

src.osm_configurator.model.project.calculation.geo_data_phase module

`class GeoDataPhase`

Bases: *`ICalculationPhase`*

This Phase is responsible for three things: 1. Converting the osm_data file into the right format. 2. Splitting the osm_data file into smaller pieces. 3. If selected, removing building which are on the border of the traffic cell. For details see the method calculate().

`calculate`(*configuration_manager*)

This method does three things: 1. It splits the big input osm_data file into multiple smaller one. There are three main reason to do that - Organisational: Each file contains the osm elements of one previously defined traffic cell. This is more organized. - Parallelization: Splitting the file into multiple smaller files allows, for better parallelization, since every thread/process can work with one file. - RAM usage: RAM capacity is limited. We can't load one big file into the memory at once, so we need to split up the file. 2. After that it converts the osm data files into files with the ".pbf" osm data file format, which is done since the library we use internally uses ".pbf" formats. 3. If the option "buildings on edge are in" didn't get selected. It removes all buildings which are on the edge/border.

Parameters
`configuration_manager` (`configuration_manager.ConfigurationManager`) – The object containing all the configuration needed for an execution.

Returns
The state of the calculation after this phase finished its execution or failed trying so.

Return type
calculation_state_enum.CalculationState

src.osm_configurator.model.project.calculation.osm_file_converter module

`class OSMFileConverter`(*file_path*)

Bases: `object`

This class handles osm file conversion, it is used to transform an osm data file from one format to the others. For more on the different file format check "calculation.OSMFileFormat" out.

`__init__`(*file_path*)

Creates a new instance of "OSMFileConverter".

Parameters
`file_path` (*`pathlib.Path`*) – The path pointing towards the file which format we want to transform into another format.

`convert_file`(*data_format*)

Transforms an osm file format into another osm file format. Allowed format: ".pbf", ".osm.bz2", ".osm".

Parameters
`data_format` (`osm_file_format_enum.OSMFileFormat`) – In which osm file format we want to transform our file into.

Returns
True if successful, otherwise false.

Return type
(bool)

src.osm_configurator.model.project.calculation.osm_file_format_enum module

class OSMFileFormat(*value*)

Bases: `Enum`

This enum describes the different osm file formats we use for file_conversion. For more information on the osm file formats check this out: https://wiki.openstreetmap.org/wiki/OSM_file_formats.

PBF = '.pbf'

BZ2 = '.osm.bz2'

OSM = '.osm'

get_file_extension()

Getter for the file extension of an enum type.

Returns
The File extension the osm file format uses.

Return type
(str)

src.osm_configurator.model.project.calculation.reduction_phase module

class ReductionPhase

Bases: *ICalculationPhase*

This calculation phase is responsible for reducing bigger OSM-elements on single coordinates and for generating the values of the attributes for alle OSM-elements. For details see the method calculate().

calculate(*configuration_manager*)

Reduces OSM-elements on single points and calculates their attributes. The calculation phase reads the data of the previous calculation phase. OSM-elements that are not just a single node, must be reduced on one coordinate. For that the centre of the given shape is calculated and set as the new coordinate. This calculation phase does also calculate the attributes of every OSM-element. There is no generic form for calculation attributes, every attribute has an individual calculation. If a method of calculation is not possible or if the user turned it off, the value of the attributes is defined by the default value list of the category. The value is given by the highest priority entry of the default value list, that matches the osm-element. After the calculations are done, the results are stored on the hard-drive.

Parameters
configuration_manager (`configuration_manager.ConfigurationManager`) – The object containing all the configuration required for an execution.

Returns
The state of the calculation after this phase finished its execution or failed trying so.

Return type
calculation_state_enum.CalculationState

src.osm_configurator.model.project.calculation.split_up_files module

class SplitUpFile(*file_path*)

Bases: `object`

This class is responsible to split up osm-data files, into multiple smaller osm-data files. This is useful since an osm-data file loaded into the ram can be bigger than the capacity of the RAM.

__init__(*file_path*)

Creates a new instance of "SplitUpFile".

Parameters
file_path (*pathlib.Path*) – The path pointing towards the osm_data file we want to split.

split_up_files(*coordinates*)

This method splits up the file into multiple smaller ones based on the coordinates it receives.

Parameters
coordinates (*List[shapely.Polygon]*) – A list of polygon. Each polygon ist the bounding box of one traffic cell we want to split up.

Returns
True if successful, otherwise false.

Return type
(bool)

src.osm_configurator.model.project.calculation.tag_filter_phase module

class TagFilterPhase

Bases: *ICalculationPhase*

This calculation phase is responsible for sorting OSM-elements into their corresponding categories. For details see the method calculate().

calculate(*configuration_manager*)

Sorts OSM-elements into their corresponding categories. Firstly this method reads in the OSM-files of the previously executed calculation phase. Every category has defined a tag filter in the configuration phase. The OSM-Elements are now sorted into the categories, depending on whether they do pass or do not pass the corresponding tag filters. A tag filter is defined by a black- and a whitelist. Each list is a collection of constraints of the tags of the osm-elements. An osm-element passes a tag filter, if all constraints of the whitelist are satisfied and no entry of the blacklist is satisfied. After execution the results shall be stored again on the hard-drive.

Parameters
configuration_manager (`configuration_manager.ConfigurationManager`) – The object containing all the configuration needed for an execution.

Returns
The state of the calculation after this phase finished its execution or failed trying so.

Return type
calculation_state_enum.CalculationState

src.osm_configurator.model.project.calculation.traffic_cell module

`class TrafficCell`

Bases: `object`

A TrafficCell is an object which describes an area on the earth, it has a name and a bounding box which entail the described area from the name.

Examples

The “Karlsruher Schloss” is a trafficCell with the name=”Karlsruher Schloss” and its bounding box being a polygon of coordinates(latitude, longitude) which entail the area of the “Karlsruher Schloss”.

`__init__()`

Creates a new instance of the TrafficCell.

`get_name()`

Getter for the name of the TrafficCell.

Returns
The name of the TrafficCell.

Return type
str

`get_bounding_box()`

Getter for the bounding box of the TrafficCell.

Returns
A list of coordinates that describe the bounding box of the TrafficCell.

Return type
shapely.Polygon

Module contents

src.osm_configurator.model.project.configuration package

Submodules

src.osm_configurator.model.project.configuration.aggregation_configuration module

`class AggregationConfiguration`

Bases: `object`

This class manages the different aggregation methods stored in an enum. Therefore, it activates and deactivates those methods. Activating means that this aggregation methods should be used during the aggregation phase in the calculation. Deactivating means the opposite.

`__init__()`

Creates a new instance of the AggregationConfiguration.

get_all_aggregation_methods()

Gives back a List of all possible aggregation methods.

Returns
A list containing all aggregation methods.

Return type
list[*aggregation_method_enum.AggregationMethod*]

is_aggregation_method_active(*method*)

Checks, if a given aggregation method is active.

Parameters
method (aggregation_method_enum.AggregationMethod) – The method, which is to be checked.

Returns
True if the aggregation method is active, otherwise false.

Return type
bool

set_aggregation_method_active(*method*, *active*)

Changes the aggregation method from active to inactive and vice versa. If an already active aggregation method should be activated, it stays active. The same applies to inactive aggregation methods, which should be deactivated.

Parameters

- **method** (aggregation_method_enum.AggregationMethod) – The method, which state should be changed.
- **active** (*bool*) – This is the new state of the aggregation method.

Returns
True if changing the state works, otherwise false.

Return type
bool

src.osm_configurator.model.project.configuration.attractivity_attribute module

class AttractivityAttribute(*attractivity_attribute_name*, *attractivity_attribute_list*, *base_attractivity*)

Bases: object

AttractivityAttribute models a single Attractivity Attributes to its factors. Each AttractivityAttribute consists of the following elements: - A name, which describes the AttractivityAttribute - A List of attributes factor pairs, which describe the attractivity attribute - A base factor

__init__(*attractivity_attribute_name*, *attractivity_attribute_list*, *base_attractivity*)

Creates a new instance of a "AttractivityAttribute" class.

Parameters

- **attractivity_attribute_name** (*str*) – The name of the Attractivity Attributes
- **attractivity_attribute_list** (*List[(*attribute_enum.Attribute*, float)]*) – A list of attributes each having its own factor.
- **base_attractivity** (*float*) – The base attractivity value.

Examples

An example for attractivity_attribute_list: [(AREA, 1.0), (NUMER_OF_FLOOR, 2.0), (GROUND_AREA, 6.9)]

`get_attractivity_attribute_name()`

Getter for attractivity attribute name.

Returns
The attractivity attribute name.

Return type
str

`set_attractivity_attribute_name(`*name*`)`

Setter for the attractivity attribute name.

Parameters
`name` (*str*) – name of the attractivity attribute.

Returns
true, if the name was successfully set, false otherwise

Return type
bool

`get_attractivity_attribute_list()`

Getter for the list of attributes and factors.

Returns
The list of attribute factor pairs.

Return type
list[(*attribute_enum.Attribute*, float)]

`set_attractivity_attribute_list(`*attractivity_attribute_list*`)`

Setter for the list of attributes and factors.

Parameters
`attractivity_attribute_list` (*list[(attribute_enum.Attribute, float)]*) – A list of attribute factor pairs we want to set as the new list of attributes and factors.

`get_base_factor()`

Getter for the base factor.

Returns
the base factor

Return type
float

`set_base_factor(`*new_base_factor*`)`

Setter for the base factor.

Parameters
`new_base_factor` (*float*) – New value for the base factor

Returns
true if the base factor eas successful set, false else

Return type
bool

src.osm_configurator.model.project.configuration.attribute_enum module

`class Attribute(value)`

Bases: `Enum`

This enum provides a list of Attributes, the DefaultValueEntry and AttractivityAttributes can use. If you are interested how exactly these Attributes get used checkout AttractivityPhase.

`PROPERTY_AREA = 'Property Area'`

The area of the property of the osm-element

`NUMER_OF_FLOOR = 'Number of Floors'`

the number of floors the osm element has

`FIRST_FLOOR_AREA = 'Floor Area'`

the area that the first floor has

`get_name()`

Getter for the name of the enum type.

Returns
Name of the Phase.

Return type
(str)

src.osm_configurator.model.project.configuration.calculation_method_of_area_enum module

`class CalculationMethodOfArea(value)`

Bases: `Enum`

Enum Provides Calculation Method of the Area.

`CALCULATE_SITE_AREA = 'Calculate Site Area'`

`CALCULATE_BUILDING_AREA = 'Calculate Building Area'`

`abstract get_calculation_method()`

Getter for the name of the Calculation Method.

Returns
The name of the Calculation Method.

Return type
str

src.osm_configurator.model.project.configuration.category module

`class Category`

Bases: `object`

Represents a category. A category is a collection of configurations for the calculation process. A category defines which OSM-elements are contained by it with a white- and a blacklist. All configurations of the category do only affect does OSM-elements.

`__init__()`

Creates a new instance of a "Category" class.

`is_active()`

Checks if value "active" is set.

Returns
True if active, false if inactive.

Return type
bool

`activate()`

Sets the active-value to True.

Returns
True, if value was set correctly, False if value was already True.

Return type
bool

`deactivate()`

Sets the active-value to False.

Returns
True, if value was set correctly, False if value was already False.

Return type
bool

`get_whitelist()`

Getter for the whitelist of the category.

Returns
List containing all whitelist values of the class.

Return type
list[str]

`set_whitelist(`*new_whitelist*`)`

Changes the old whitelist to a new one.

Parameters
`new_whitelist` (*`list[str]`*) – value for the new whitelist.

Returns
True, if the whitelist was overwritten successfully, else False.

Return type
bool

`get_blacklist()`

Getter for the blacklist of the category.

Returns
list[str]: list containing all blacklist attributes of the class.

`set_blacklist(`*new_blacklist*`)`

Overwrites the old Blacklist with a new value.

Parameters
`new_blacklist` (*`list[str]`*) – new value for the blacklist.

Returns
True, if the blacklist was overwritten successfully, else False.

Return type
bool

`get_category_name()`

Getter for the category name.

Returns
name of the category.

Return type
str

`set_category_name(`*new_category_name*`)`

Overwrites the old category_name.

Parameters
`new_category_name` (*`str`*) – new value for the category_name.

Returns
True, if the overwriting process concluded successfully, else False.

Return type
bool

`get_calculate_area()`

This says if the area of the category should be calculated or not.

Returns
true if it should get calculated, false if not.

Return type
bool

`get_calculation_method_of_area()`

Getter for the calculated area method.

Returns
The method with which we calculate the area.

Return type
calculation_method_of_area_enum.CalculationMethodOfArea

`set_calculation_method_of_area(`*new_calculate_area*`)`

Overwrites current calculate_area with the given value.

Parameters
`new_calculate_area` (*`bool`*) – new value that will overwrite the existing value.

`get_calculate_floor_area()`

This says if the floor area should be calculated or not.

Returns
true if the floor are should be calculated, otherwise false.

Return type
bool

`set_calculate_floor_area(`*new_calculate_floor_area*`)`

Overwrites the existing instance of calculate_floor_area.

Parameters
new_calculate_floor_area (*bool*) – new value for calculate_floor_are.

Returns
True, if the overwriting process was successful, else false.

Return type
bool

`get_strictly_use_default_values()`

This says if in the calculation we should strictly use the default values.

Returns
value of strictly_use_default_values.

Return type
bool

`set_strictly_use_default_values(`*new_strictly_use_default_values*`)`

Overwrites the already existing value of strictly_use_default_values.

Parameters
new_strictly_use_default_values (*bool*) – new value for strictly_use_default_values.

Returns
True if the overwriting process was successful, else False.

`get_attractivity_attributes()`

Getter for the AttractivityAttributes of the category.

Returns
List of all used attractivity attributes

Return type
list[*attractivity_attribute.AttractivityAttribute*]

`add_attractivity_attribute(`*new_attractivity_attribute*`)`

Adds a new attractivity attribute to the list.

Parameters
new_attractivity_attribute (`attractivity_attribute.AttractivityAttribute`) – new attractivityAttribute that will be added.

Returns
True, if the attribute was added successfully, else False.

Return type
bool

`remove_attractivity_attribute`(*attractivity_attribute*)

Removes an already existing attribute from the list.

Parameters
`attractivity_attribute` (`attractivity_attribute.AttractivityAttribute`) – attractivity attribute that will be removed from the list.

Returns
True, if the element was removed, else False.

`get_default_value_list`()

Getter for the default values of the category.

Returns
List of all used default values.

Return type
list[*DefaultValueEntry*]

`add_default_value_entry`(*new_default_value_entry*)

Adds a new value to the default_value_entry list.

Parameters
`new_default_value_entry` (`DefaultValueEntry`) – element to add.

Returns
True, if element was added successfully, else False

Return type
bool

`remove_default_value_entry`(*default_value_entry*)

Removes an already existing element from the default_value_entry list.

Parameters
`default_value_entry` (`default_value_entry.DefaultValueEntry`) – value that will be removed.

Returns
True, if the element was removed successfully, else False.

Return type
bool

`move_default_value_entry_up`(*default_value_entry*)

Moves an already existing default value from the list one element up.

Parameters
`default_value_entry` (`default_value_entry.DefaultValueEntry`) – element from the list, that will be incremented by one.

Returns
True, if the change was successful, else False.

Return type
bool

`move_default_value_entry_down`(*default_value_entry*)

Moves an already existing default value from list one element down.

Parameters
`default_value_entry` (`default_value_entry.DefaultValueEntry`) – element from the list, that will be decremented by one.

Returns
True, if the change was successful, else false.

Return type
bool

src.osm_configurator.model.project.configuration.category_manager module

`class CategoryManager`(*categories*)

Bases: `object`

Category Manager holds a list of categories and changes them according to the given needs.

`__init__`(*categories*)

Constructor of the class.

Parameters
`categories` (`Category`) – Starting list of categories.

`get_category`(*index*)

Gets a category based on the index.

Parameters
`index` (*int*) – Index in the categories-list, that will be returned.

Returns
The Category we wanted.

Return type
category.Category

`get_categories`()

Getter for all the Categories.

Returns
List of the chosen categories.

Return type
list[*Category*]

`create_category`(*new_category*)

Creates a new category, that will be empty.

Returns
The newly created category.

Return type
category.Category

`remove_category`(*category*)

Removes the given category from the categories list, if element is inside the List.

Parameters
`category` (`Category`) – Category that will be removed.

Returns
True, if the element was removed correctly, else false.

Return type
bool

`override_categories`(*new_category_list*)

Overwrites the list of categories with the given list, if both lists are not identical.

Parameters
new_category_list (*list[Categories]*) – List of categories, that will overwrite the already existing list.

Returns
True, if the replacement was successful, else False.

Return type
bool

`merge_categories`(*category_input_list*)

Merges the existing category list with the given list if both lists are not identical.

Parameters
category_input_list (*list[Category]*) – New list of categories that will be merged into the existing list.

Returns
True, if the merging was successful, else False.

Return type
bool

src.osm_configurator.model.project.configuration.configuration_manager module

`class ConfigurationManager`(*active_project_path*)

Bases: `object`

This class job is to manage the configurations of the OSM data, aggregation, cut-out and categories. It also makes this information available to the calculation

`__init__`(*active_project_path*)

Creates a new instance of the ConfigurationManager.

Parameters
active_project_path (*pathLib.Path*) – The path pointing towards the project folder.

`get_osm_data`()

Gives back the path pointing towards the OSM data file.

Returns
The path pointing towards the OSM data.

Return type
pathlib.Path

`set_osm_data`(*osm_data*)

Edits the path pointing towards the OSM data file.

Parameters
osm_data (*pathlib.Path*) – The new path towards the osm data file.

Returns
True if changing the path works, otherwise false.

Return type
bool

`get_all_aggregation_methods()`

Gives back a List of all possible aggregation methods.

Returns
A list containing all aggregation methods.

Return type
list[*aggregation_method_enum.AggregationMethod*]

`is_aggregation_method_active(`*method*`)`

Checks, if a given aggregation method is active.

Parameters
`method` (`aggregation_method_enum.AggregationMethod`) – The method, which is to be checked.

Returns
True if the aggregation method is active, otherwise false.

Return type
bool

`set_aggregation_method_active(`*method*, *active*`)`

Changes the aggregation method from active to inactive and vice versa. If an already active aggregation method should be activated, it stays active. The same applies to inactive aggregation methods, which should be deactivated.

Parameters

- **`method`** (`aggregation_method_enum.AggregationMethod`) – The method, which state should be changed.
- **`active`** (*`bool`*) – This is the new state of the aggregation method.

Returns
True if changing the state works, otherwise false.

Return type
bool

`get_cut_out_mode()`

Gives back the used cut-out mode.

Returns
The used cut-out mode.

Return type
cut_out_mode_enum.CutOutMode

`set_cut_out_mode(`*new_cut_out_mode*`)`

Changes the cut-out mode used during the reduction phase in the calculation.

Parameters
`new_cut_out_mode` (`cut_out_mode_enum.CutOutMode`) – The new cut-out mode for the calculation.

Returns
True if changing the cut-out mode works, otherwise false.

Return type
bool

`get_cut_out_path()`

Gives back the path pointing towards the cut-out file.

Returns
The path pointing towards the cut-out.

Return type
pathlib.Path

`set_cut_out_path`(*path*)

Changes the path pointing towards the cut-out file.

Parameters
path (*pathlib.Path*) – The new path.

Returns
True if changing the cut-out path works, otherwise false.

Return type
bool

`get_category`(*index*)

Gets a category based on the index.

Parameters
index (*int*) – Index in the categories-list, that will be returned.

Returns
The Category we wanted.

Return type
category.Category

`get_categories()`

Getter for all the Categories.

Returns
List of the chosen categories.

Return type
list[*Category*]

`create_category()`

Creates a new category, that will be empty.

Returns
The newly created category.

Return type
category.Category

`remove_category`(*category*)

Removes the given category from the categories list, if element is inside the List.

Parameters
category (`Category`) – Category that will be removed.

Returns
True, if the element was removed correctly, else false.

Return type
bool

`override_categories`(*new_category_list*)

Overwrites the list of categories with the given list, if both lists are not identical.

Parameters
`new_category_list` (*list[Categories]*) – List of categories, that will overwrite the already existing list.

Returns
True, if the replacement was successful, else False.

Return type
bool

`merge_categories`(*category_input_list*)

Merges the existing category list with the given list if both lists are not identical.

Parameters
`category_input_list` (*list[Category]*) – New list of categories that will be merged into the existing list.

Returns
True, if the merging was successful, else False.

Return type
bool

`get_active_project`()

This method gives back the active project.

Returns
The path pointing towards the current project folder.

Return type
pathlib.Path

`get_is_data_downloaded`()

Gives back if data in DownloadData is downloaded.

Returns
True if data is downloaded, otherwise false.

Return type
bool

src.osm_configurator.model.project.configuration.cut_out_configuration module

`class CutOutConfiguration`

Bases: `object`

This class job is to store the cut-out mode and the cut-out file path. Both are required during the reduction-phase in the calculation.

`__init__`()

Creates a new instance of the "CutOutConfiguration" class.

`get_cut_out_mode()`

Gives back the used cut-out mode.

Returns
The used cut-out mode.

Return type
cut_out_mode_enum.CutOutMode

`set_cut_out_mode(`*new_cut_out_mode*`)`

Changes the cut-out mode used during the reduction phase in the calculation.

Parameters
new_cut_out_mode (`cut_out_mode_enum.CutOutMode`) – The new cut-out mode for the calculation.

Returns
True if changing the cut-out mode works, otherwise false.

Return type
bool

`get_cut_out_path()`

Gives back the path pointing towards the cut-out file.

Returns
The path pointing towards the cut-out.

Return type
pathlib.Path

`set_cut_out_path(`*path*`)`

Changes the path pointing towards the cut-out file.

Parameters
path (*pathlib.Path*) – The new path.

Returns
True if changing the cut-out path works, otherwise false.

Return type
bool

src.osm_configurator.model.project.configuration.cut_out_mode_enum module

`class CutOutMode(`*value*`)`

Bases: `Enum`

The job of this enum is to store the different cut-out-modes used in the reduction during the calculation. We differentiate on, if we should include building which are on the edge/border, this mean partially inside the traffic cell, or not.

`BUILDINGS_ON_EDGE_ACCEPTED = 'Buildings on edge are accepted'`

`BUILDINGS_ON_EDGE_NOT_ACCEPTED = 'Building on the edge are not accepted'`

`get_name()`

Getter for the name of the enum type.

Returns
the name of the enum

Return type
str

src.osm_configurator.model.project.configuration.default_value_entry module

class DefaultValueEntry(*tag*, *attribute_default_values*)

Bases: `object`

DefaultValueEntry stores a default value for every attribute. Default values can be set and read.

`__init__`(*tag*, *attribute_default_values*)

Constructor of the class, creates an empty DefaultValueEntry with 0 for all the factor values.

`get_default_value_entry_tag`()

Returns the tag associated with this default value entry :returns: The tag of this entry :rtype: str

`set_tag`(*new_tag*)

Sets a new value for a given tag

Parameters
new_tag (*str*) – value for overwriting the current tag, must be a valid OSM-tag

Returns
true if the overwriting process was successful, else false

Return type
bool

`set_attribute_default`(*attribute*, *value*)

Sets the default value of an attribute

Parameters

- **attribute** (`attribute_enum.Attribute`) – Attribute whose value will be overwritten
- **value** (*float*) – new default value

Returns
true, if overwriting process was successful, else false

Return type
bool

`get_attribute_default`(*attribute*)

Gets the default value of a certain attribute

Parameters
attribute (`attribute_enum.Attribute`) – Attribute whose value is searched for

Returns
The default value of the attribute

Return type
float

src.osm_configurator.model.project.configuration.download_data module

`class DownloadData`

Bases: `object`

This class manages the download of OSM data depending on a list of coordinates.

`__init__()`

Creates a new instance of the DownloadData.

`download_data`(*coordinates*)

Downloads the OSM data which the coordinates dictate.

Parameters
`coordinates` (*shapely.Polygon*) – The new area, which should be downloaded

Returns
True when the download works, otherwise false.

Return type
bool

src.osm_configurator.model.project.configuration.osm_data_configuration module

`class OSMDataConfiguration`

Bases: `object`

The job of the OSMDataConfiguration is to store the path pointing towards the OSM data file.

`__init__()`

Creates a new instance of the "OSMDataConfiguration" class.

`get_osm_data()`

Gives back the path pointing towards the OSM data file.

Returns
The path pointing towards the OSM data.

Return type
pathlib.Path

`set_osm_data`(*osm_data*)

Edits the path pointing towards the OSM data file.

Parameters
`osm_data` (*pathlib.Path*) – The new path towards the osm data file.

Returns
True if changing the path works, otherwise false.

Return type
bool

`download_data`(*coordinates*)

Downloads the OSM data which the coordinates dictate.

Parameters
`coordinates` (*shapely.Polygon*) – The new area, which should be downloaded

Returns
True when the download works, otherwise false.

Return type
bool

Module contents

Submodules

src.osm_configurator.model.project.active_project module

`class ActiveProject`(*project_folder*, *is_newly_created*)

Bases: `object`

This class job is to manage the active project the user is working on. Whereby an active project, a project is that got selected by the user in the project selected screen or created.

`__init__`(*project_folder*, *is_newly_created*)

Creates a new instance of the ActiveProject. In this process it creates the ConfigurationManager and also differentiate between the case that the project is new or loaded. In the case of an existing project it calls the ProjectLoader, otherwise it creates a new project.

Parameters

- **`project_folder`** (*`pathlib.Path`*) – This is path pointing towards the folder, where the project is saved.
- **`is_newly_created`** (*`bool`*) – This argument is true if the project is newly created, otherwise false.

`create`(*name*, *description*)

This method creates a new project and adds a name and a description to it.

Parameters

- **`name`** (*`str`*) – The name of the project.
- **`description`** (*`str`*) – A description of the project.

Returns
True if creating the project works, otherwise false.

Return type
bool

`get_last_step`()

This method is there so that the user can continue working in the same phase in an existing project where he previously stopped.

Returns
The last phase the user was working on.

Return type
config_phase_enum.ConfigPhase

`start_calculation`(*calculation_phase*)

This method is to start the calculation (after the configuration is finished).

Parameters
`calculation_phase` (`calculation_phase_enum.CalculationPhase`) – The calculation phase, where the calculation shall start.

Returns
The calculation state where the calculation started. Can be an error state, so signify an error, that prevented the start of the calculation.

Return type
calculation_state_enum.CalculationState

`get_project_path()`

This method is to give back the path pointing towards the project folder.

Returns
The path pointing towards the project folder.

Return type
pathlib.Path

`get_osm_data()`

Gives back the path pointing towards the OSM data file.

Returns
The path pointing towards the OSM data.

Return type
pathlib.Path

`set_osm_data`(*osm_data*)

Edits the path pointing towards the OSM data file.

Parameters
`osm_data` (*pathlib.Path*) – The new path towards the osm data file.

Returns
True if changing the path works, otherwise false.

Return type
bool

`get_all_aggregation_methods()`

Gives back a List of all possible aggregation methods.

Returns
A list containing all aggregation methods.

Return type
list[*aggregation_method_enum.AggregationMethod*]

`is_aggregation_method_active`(*method*)

Checks, if a given aggregation method is active.

Parameters
`method` (`aggregation_method_enum.AggregationMethod`) – The method, which is to be checked.

Returns
True if the aggregation method is active, otherwise false.

Return type
bool

`set_aggregation_method_active`(*method*, *active*)

Changes the aggregation method from active to inactive and vice versa. If an already active aggregation method should be activated, it stays active. The same applies to inactive aggregation methods, which should be deactivated.

Parameters

- `method` (`aggregation_method_enum.AggregationMethod`) – The method, which state should be changed.
- `active` (*`bool`*) – This is the new state of the aggregation method.

Returns
True if changing the state works, otherwise false.

Return type
bool

`get_cut_out_mode`()

Gives back the used cut-out mode.

Returns
The used cut-out mode.

Return type
cut_out_mode_enum.CutOutMode

`set_cut_out_mode`(*new_cut_out_mode*)

Changes the cut-out mode used during the reduction phase in the calculation.

Parameters
`new_cut_out_mode` (`cut_out_mode_enum.CutOutMode`) – The new cut-out mode for the calculation.

Returns
True if changing the cut-out mode works, otherwise false.

Return type
bool

`get_cut_out_path`()

Gives back the path pointing towards the cut-out file.

Returns
The path pointing towards the cut-out.

Return type
pathlib.Path

`set_cut_out_path`(*path*)

Changes the path pointing towards the cut-out file.

Parameters
`path` (*`pathlib.Path`*) – The new path, towards a cut-out file.

Returns
True if changing the cut-out path works, otherwise false.

Return type
bool

`get_category`(*index*)

Gets a category based on the index.

Parameters
: `index` (*int*) – Index in the categories-list, that will be returned.

Returns
: The Category we wanted, NONE if the index is out of bounds of the list.

Return type
: *category.Category*

`get_categories`()

Getter for all the Categories.

Returns
: List of the chosen categories.

Return type
: list[*category.Category*]

`create_category`()

Creates a new category, that will be empty.

Returns
: The newly created category.

Return type
: *category.Category*

`remove_category`(*category*)

Removes the given category from the categories list, if element is inside the List.

Parameters
: `category` (`category.Category`) – Category that will be removed.

Returns
: True, if the element was removed correctly, else false.

Return type
: bool

`override_categories`(*new_category_list*)

Overwrites the list of categories with the given list, if both lists are not identical.

Parameters
: `new_category_list` (*list[*`category.Category`*]*) – List of categories, that will overwrite the already existing list.

Returns
: True, if the replacement was successful, else false.

Return type
: bool

`merge_categories`(*category_input_list*)

Merges the existing category list with the given list if both lists are not identical. If two categories conflict in their name, the newer category will be used.

Parameters
: `category_input_list` (*list[*`category.Category`*]*) – New list of categories that will be merged into the existing list.

Returns
True, if the merging was successful, else False.

Return type
bool

`create_map`(*cut_out*)

This method to create a map from to given cut-out.

Parameters
`cut_out` (`cut_out_configuration.CutOutConfiguration`) – The cut-out configuration from which the map should be created.

Returns
True if creating the map works, otherwise false.

Return type
bool

`create_boxplot`(*data*)

This method is to visualize the data by creating a boxplot. It is used to visualize the calculated end result via a boxplot.

Parameters
`data` (*`matplotlib.axes.Axes`*) – A plot of the data which we want to visualize.

Returns
True if creating the boxplot works, otherwise false.

Return type
bool

`get_location`()

Getter for the location of the active project on the disk.

Returns
The location of the active project

Return type
pathlib.Path

`set_name`(*new_name*)

This method changes the name of the project.

Parameters
`new_name` (*`str`*) – The new name of the project

Returns
true if change was successful, false else

Return type
bool

`get_name`()

This method returns the name of the project.

Returns
name of the project

Return type
str

`set_description`(*new_description*)

This method changes the description of the project.

Parameters
`new_description` (*str*) – The new description of the project

Returns
true if change successful, false else

Return type
bool

`get_description`()

This method returns the description of the project.

Returns
The description of the project

Return type
str

`export_project`(*path*)

Exports the whole project to the given path.

Parameters
`path` (*pathlib.Path*) – The path where the project shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

`export_configuration`(*path*)

Exports the configuration to the given path. More specific, exports all the categories and their configurations, to the given path.

Parameters
`path` (*pathlib.Path*) – The path where the configurations shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

`export_calculation`(*path*)

Exports the results of the calculation to the given path. Whereby the calculation are a folder with all the different results from each calculation step in it.

Parameters
`path` (*pathlib.Path*) – The path where the results of the calculation shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

`export_map`(*path*)

Exports an HTML-Data with the map in it, to the given path.

Parameters
`path` (*pathlib.Path*) – The path, where the map shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

src.osm_configurator.model.project.config_phase_enum module

class ConfigPhase(*value*)

Bases: `Enum`

This enum stores the different phases of the configuration and is used to restores the last step the user was working on.

DATA_CONFIG_PHASE = 'Data Configuration Phase'

CATEGORY_CONFIG_PHASE = 'Category Configuration Phase'

REDUCTION_CONFIG_PHASE = 'Reduction Configuration Phase'

ATTRACTIVITY_CONFIG_PHASE = 'Attractivity Configuration Phase'

AGGREGATION_CONFIG_PHASE = 'Aggregation Configuration Phase'

CALCULATION_CONFIG_PHASE = 'Calculation Configuration Phase'

get_name()

Getter for the name of the phase.

Returns
Name of the Phase.

Return type
str

src.osm_configurator.model.project.data_visualizer module

class DataVisualizer

Bases: `object`

This class job is to visualize the cut-out file or data of the project.

__init__()

Creates a new instance of the DataVisualizer.

create_map(*cut_out*)

This method to create a map from to given cut-out.

Parameters
`cut_out` (`cut_out_configuration.CutOutConfiguration`) – The cut-out configuration from which the map should be created.

Returns
True if creating the map works, otherwise false.

Return type
bool

`create_boxplot`(*data*)

This method is to visualize the data by creating a boxplot. It is used to visualize the calculated end result via a boxplot.

Parameters
data (`matplotlib.axes.Axes`) – A plot of the data which we want to visualize.

Returns
True if creating the boxplot works, otherwise false.

Return type
bool

src.osm_configurator.model.project.export module

`class Export`(*project*)

Bases: `object`

This class provides different export features based on the current project. Whereby exporting means, saving data from the currently active project somewhere else on the system on the disk.

`__init__`(*project*)

Creates a new instance of the "Export" class.

Parameters
project (`project.ActiveProject`) – The project to make exports from

`export_project`(*path*)

Exports the whole project to the given path.

Parameters
path (`pathlib.Path`) – The path where the project shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

`export_configuration`(*path*)

Exports the configuration to the given path. More specific, exports all the categories and their configurations, to the given path.

Parameters
path (`pathlib.Path`) – The path where the configurations shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

`export_calculation`(*path*)

Exports the results of the calculation to the given path. Whereby the calculation are a folder with all the different results from each calculation step in it.

Parameters
`path` (*Path*) – The path where the results of the calculation shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

`export_map`(*path*)

Exports an HTML-Data with the map in it, to the given path.

Parameters
`path` (*Path*) – The path, where the map shall be exported to

Returns
true, if export was successful, otherwise false.

Return type
bool

src.osm_configurator.model.project.project_io_handler module

`class ProjectIOHandler`(*active_project*)

Bases: `object`

This class handles the I/O of a project. This includes: - loading a project from disk into memory that the user selected in the main menu - creating a project on the disk, if the user selected that

`__init__`(*active_project*)

Creates a new instance of the ProjectLoader. Therefore, it gets the current active project, which should be loaded if not newly created.

Parameters
`active_project` (`active_project.ActiveProject`) – The project the ProjectLoader shall load.

`build_project`(*path*)

This method is to build the given project. To do this it reads out the configurations and builts a folder structure on the disk.

Parameters
`path` (*pathlib.Path*) – The path pointing towards the project folder.

Returns
True if creating the project works, otherwise false.

Return type
bool

`load_project`(*path*)

Loads a project in from the disk into the memory.

Parameters
`path` (*pathlib.Path*) – The path pointing towards the project folder.

Returns
True if creating the project works, otherwise false.

Return type
bool

src.osm_configurator.model.project.project_saver module

`class ProjectSaver`(*active_project*)

Bases: `object`

The ProjectSave is responsible for saving the internal representation of the project onto the disk.

`__init__`(*active_project*)

Creates a new instance of the ProjectSaver. Therefore, it gets the current active project, which should be loaded if not newly created.

Parameters

`active_project` (`active_project.ActiveProject`) – The project the ProjectSaver shall load.

`save_project`(*path*)

Stores all the configurations of the project. The information about the configuration of the project are stored to the disk.

Parameters

`path` (*`pathlib.Path`*) – The path pointing towards the project folder. The data will be stored here

Returns

True, if the project was stored successfully, False, if an error occurred.

Return type

bool

src.osm_configurator.model.project.project_settings module

`class ProjectSettings`(*location*, *project_name*, *description*)

Bases: `object`

This class saves all the different settings of a project and provides methods to view and change them.

`__init__`(*location*, *project_name*, *description*)

This method creates a new ProjectSettings class with the given settings.

Parameters

- **`location`** (*`pathlib.Path`*) – The location, the project is stored
- **`project_name`** (*`str`*) – The name of the project
- **`description`** (*`str`*) – A description of the project

`get_location`()

Getter for the location of the Project on the disk.

Returns

The location of the project

Return type

pathlib.Path

`set_location`(*new_location*)

This method changes the location where the project will be stored.

Parameters
`new_location` (*pathlib.Path*) – The new location for the project

Returns
true, if location change was successful, false else

Return type
bool

`set_name`(*new_name*)

This method changes the name of the project.

Parameters
`new_name` (*str*) – The new name of the project

Returns
true if change was successful, false else

Return type
bool

`get_name`()

This method returns the name of the project.

Returns
name of the project

Return type
str

`set_description`(*new_description*)

This method changes the description of the project.

Parameters
`new_description` (*str*) – The new description of the project

Returns
true if change successful, false else

Return type
bool

`get_description`()

This method returns the description of the project.

Returns
The description of the project

Return type
str

Module contents

Module contents

src.osm_configurator.view package

Subpackages

src.osm_configurator.view.popups package

Submodules

src.osm_configurator.view.popups.alert_pop_up module

`class` **`AlertPopUp`**(*message*)

Bases: `CTkToplevel`

This class creates popups, that will pop up in front of the GUI. This instance is an Alert-PopUp. It provides a message and one 'OK' button, to close the PopUp again.

`__init__`(*message*)

This constructor will create an AlertPopUp. It will provide the given message and an 'OK' button to close the PopUp again.

Parameters

`message` (*`str`*) – The message that will be shown by the AlertPopUp

src.osm_configurator.view.popups.yes_no_pop_up module

`class` **`YesNoPopUp`**(*message*, *func*)

Bases: `CTkToplevel`

This class creates PopUps, that will pop up in front of the GUI. This instance is a YesNoPopUp: It will provide a message, an 'OK' and an 'Cancel' button. Pressing a button will return the information back to the creating instance and close the PopUp.

`__init__`(*message*, *func*)

This constructor will create an YesNoPopUp, that will show the given message, as well as an 'OK' and 'Cancel' button. If one of the buttons is pressed or the PopUp closed, it will send a message back via the given function, what button had been pressed. If 'OK' has been pressed, the given function will be called with a boolean = true. If 'Cancel' has been pressed, or the PopUp was closed otherwise, the given function will be called with a boolean = false.

Parameters

- **`message`** (*`str`*) – The message to be shown in the PopUp.
- **`func`** (*`Callable`*) – A Function that takes one Boolean and has no return, for the PopUp to send a message back.

Module contents

src.osm_configurator.view.states package

Submodules

src.osm_configurator.view.states.main_window module

`class MainWindow`(*control*)

Bases: `object`

This class provides the GUI, the user will be working on. It is made dynamic and can change between different frames, to show different information and buttons to the user. Its job is to just show the frames of different states and create the window the GUI will be used on.

`__init__`(*control*)

This method creates a MainWindow with a connection to the given control.

Parameters

control (`control_interface.IControl`) – The control the GUI shall be working with, to get access to information on the model.

`change_state`(*last_state*, *new_state*)

This method changes from an old given state to a new given state to show on the MainWindow.

Parameters

- **last_state** (`state.State`) – The state that needs to me removed from the MainWindow
- **new_state** (`state.State`) – The state that shall be shown by the MainWindow

Returns

true, if the state change was successful, false if not.

Return type

bool

src.osm_configurator.view.states.positioned_frame module

`class PositionedFrame`(*frame*, *colum*, *line*, *state_manager*, *control*)

Bases: `object`

This Class gives a Frame a position, via Coordinates, to tell in what position the Frame wants to be in, when it is shown on a Window.

`__init__`(*frame*, *colum*, *line*, *state_manager*, *control*)

This method creates a PositionedFrame, which is a Frame and coordinates to its Position.

Parameters

- **frame** (`top_level_frame.TopLevelFrame`) – The frame you want to give a position
- **colum** (*int*) – The Colum the Frame shall be placed in
- **line** (*int*) – The Line the Frame shall be placed in
- **state_manager** (`state_manager.StateManager`) – The StateManager the frame will use to change states if needed

- **control** (`control_interface.IControl`) – The control that a frame shall call, if it needs access to the model

get_frame()

This method returns the frame this PositionedFrame holds.

Returns
The Frame this PositionedFrame holds

Return type
(*top_level_frame.TopLevelFrame*)

get_column()

This method Returns the column the frame is placed in.

Returns
The Column the Frame is placed in.

Return type
(int)

get_line()

This method returns the line the frame is placed in.

Returns
The Line the frame is placed in

Return type
(int)

src.osm_configurator.view.states.state module

class State(*active_frames*, *own_state_name*, *default_left*, *default_right*)

Bases: `object`

This class models a state. A State consist of - a list of frames, that shall be visible on a window, when this state is active - a default state to its right - a default state to its left All states have one state to their left and to their right, to model the basic state change. Those states can be 'none' in order so signify that there is no further right or left.

__init__(*active_frames*, *own_state_name*, *default_left*, *default_right*)

This method creates a new state, that holds the given frames, has the given state name, and has a default left and right.

Parameters

- **active_frames** (*list*[`positioned_frame.PositionedFrame`]) – A list of frames, that this state holds
- **own_state_name** (`state_name_enum.StateName`) – The name that defines this state
- **default_left** (`state_name_enum.StateName`) – The name of the state on this states left
- **default_right** (`state_name_enum.StateName`) – The name of the state on this states right

get_active_frames()

The list of frames this state holds and shall be shown on a window, if it is active.

Returns
List of frames this state holds

Return type
list[*positioned_frame.PositionedFrame*]

`get_default_left()`

The name of the state on this states left.

Returns
This states left state name

Return type
(*state_name_enum.StateName*)

`get_default_right()`

The id of the state on this states right.

Returns
This states right state name

Return type
(*state_name_enum.StateName*)

`get_state_name()`

The name of this state.

Returns
The name of this state

Return type
(*state_name_enum.StateName*)

`equals`(*state*)

Test if two states are equal. Two states are defined as equal, if their name is equal.

Parameters
`state` (`state.State`) – The state you want to know if it is equal to this one

Returns
true if the given state and this state ist equal. false if not

Return type
bool

src.osm_configurator.view.states.state_manager module

`class StateManager`(*control*, *main_window*)

Bases: `object`

This class manages the different states, that can be shown on a window. It knows what state is currently active and provides methods to change the state.

`__init__`(*control*, *main_window*)

This method creates a StateManager, that will control what state is currently active and manages the changes between states. It will create all states, as well all the frames that exist and put them in the state they belong.

Parameters

- **`control`** (`control_interface.IControl`) – The connection to the control, so the frames of each state can access the model.

- **main_window** (`main_window.MainWindow`) – The MainWindow, where the frames of the state shall be shown on.

`default_go_right()`

This method changes to the State that is the default_right state of the current one.

Returns
true, if a state change was successfully made, false if there was no state change or something went wrong

Return type
bool

`default_go_left()`

This method changes to the state that is the default_left State of the current one.

Returns
true, if a state change was successfully made, false if there was no state change or something went wrong

Return type
bool

`change_state`(*new_state*)

This method changes to the given state and deactivate the old one.

Parameters
new_state (`state_name_enum.StateName`) – The id of the new state that shall be activated.

Returns
true if state change was successful, false if not.

Return type
bool

`get_state()`

This method returns the currently active state.

Returns
the currently active state.

Return type
state.State

`close_program()`

This method closes the program and shuts the whole application down.

src.osm_configurator.view.states.state_name_enum module

`class StateName`(*value*)

Bases: `Enum`

This enum saves the different state possibilities that exist, to define a state by this enum and not, by a number. This enum gives a state a name.

`MAIN_MENU = 1`

`CREATE_PROJECT = 2`

`DATA = 3`

`CATEGORY = 4`

`REDUCTION = 5`

`ATTRACTIVITY_EDIT = 6`

`ATTRACTIVITY_VIEW = 7`

`AGGREGATION = 8`

`CALCULATION = 9`

`SETTINGS = 10`

Module contents

src.osm_configurator.view.toplevelframes package

Submodules

src.osm_configurator.view.toplevelframes.aggregation_frame module

`class AggregationFrame`(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame shows the aggregation page the user will interact on. This window provides the checkboxes to choose calculation methods and methods on how the aggregation will be calculated.

`__init__`(*state_manager*, *control*)

This method creates an AggregationFrame, that will be used to edit the aggregation method.

Parameters

- **state_manager** (`state_manager.StateManager`) – The StateManager, the frame will call, when it wants to change to another state.
- **control** (`control_interface.IControl`) – The control, the Frame will call, to get access to the model.

`activate`()

Tells the current frame to activate and collect all the data it needs.

Returns

True, if activation was successful, false else

Return type

bool

src.osm_configurator.view.toplevelframes.attractivity_edit_frame module

class AttractivityEditFrame(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame lets the user edit, create and delete attractivity attributes for the categories. Two drop-down menus will be shown: One will select the category, the other the attractivity attributes. Editing options for the attractivity attributes will be offered in a textbox to change the name. Smaller boxes provide the means of changing the factors for different attributes. Two buttons provide creation and deletion tools.

__init__(*state_manager*, *control*)

This method creates an AttractivityEditFrame, where the attractivity attributes of categories can be edited, created or be deleted.

Parameters

- **state_manager** (state_manager.StateManager) – The StateManager the frame will call, when it wants to change to another state.
- **control** (control_interface.IControl) – The control the frame will call to get access to the model.

activate()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.toplevelframes.attractivity_view_frame module

class AttractivityViewFrame(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame shows a list with all categories, their attractivity attributes and how they are calculated. This is only a visualisation and therefore a non-edit Frame.

__init__(*state_manager*, *control*)

This method creates an AttractivityViewFrame showing a lList of containing all categories, their according attractivity attributes and how they are calculated.

Parameters

- **state_manager** (state_manager.StateManager) – The StateManager the Frame will call, if it tries to switch to another atate.
- **control** (control_interface.IControl) – The control the frame will call to get access to the model.

activate()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.toplevelframes.calculation_frame module

`class CalculationFrame`(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame lets the user start the calculation from a selected phase, indicated by different buttons. Once a calculation is started, there will be a progressbar shown, the different buttons will be deactivated and the current calculation-phase will be shown. A cancel-Button is provided to stop the calculation. The CalculationFrame shows popups, if an error accures in the calculations.

`__init__`(*state_manager*, *control*)

This method creates a CalculationFrame that will let the user start the calculation and shows the calculation progress.

Parameters

- **state_manager** (`state_manager.StateManager`) – The StateManager the frame will call, if it wants to switch to another state.
- **control** (`control_interface.IControl`) – The control the frame will call to get access to the model.

`activate`()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.toplevelframes.category_frame module

`class CategoryFrame`(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame lets the user create, delete and edit categories. It shows the name of a category, as well as their black- and white-List. Categories also can be turned on and off with Checkboxes. There will also be key-recommendations be shown for the black- and white-List.

`__init__`(*state_manager*, *control*)

This method creates an CategoryFrame so the user can create, delete and edit categories.

Parameters

- **state_manager** (`state_manager.StateManager`) – The StateManager the frame will call, when it wants to change to another state.
- **control** (`control_interface.IControl`) – The control the frame will call to get access to the model.

`activate`()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.toplevelframes.create_project_frame module

class CreateProjectFrame(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame shows the project creation page to the User. A name, a description and a path for storing the project can be set here. The user can cancel the creation-process.

__init__(*state_manager*, *control*)

This method creates a CreateProjectFrame where a user can create a new project.

Parameters

- **state_manager** (`state_manager.StateManager`) – The StateManager the frame will call, if it wants to change to another State.
- **control** (`control_interface.IControl`) – The frame will call the control, to gain access to the model.

activate()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.toplevelframes.data_frame module

class DataFrame(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame lets the user edit various following Data: - Selection of the OSM-Data - Selection of the Cut-Out - Select, if buildings on the edge shall be included or not - A download button to download the OSM data after a cut-out was selected - Copy in category configurations

__init__(*state_manager*, *control*)

This method creates a DataFrame, that lets the User input data into the project.

Parameters

- **state_manager** (`state_manager.StateManager`) – The frame will call the StateManager, if it wants to switch states.
- **control** (`control_interface.IControl`) – The frame will call the control, to gain access to the model.

activate()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.toplevelframes.main_menu_frame module

`class MainMenuFrame`(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame shows the application's main menu. The user can create a new project, or load an already existing project. Projects stored in the default folder will be shown in a list and can be selected / opened.

`__init__`(*state_manager*, *control*)

This method creates a MainMenuFrame showing the MainMenu of the application.

Parameters

- **`state_manager`** (`state_manager.StateManager`) – The frame will call the StateManager, if it wants to switch states.
- **`control`** (`control_interface.IControl`) – The frame will call the control to gain access to the model.

`activate()`

Tells the current frame to activate and collect all the data it needs.

Returns

True, if activation was successful, false else

Return type

bool

src.osm_configurator.view.toplevelframes.project_foot_frame module

`class ProjectFootFrame`(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame shows two arrows on the bottom of the Window. The user can navigate the pipeline by going left or right.

`__init__`(*state_manager*, *control*)

This method creates a ProjectFootFrame, that lets the user navigate the pipeline by going left or right.

Parameters

- **`state_manager`** (`state_manager.StateManager`) – The StateManager the frame will call, if it wants to switch states.
- **`control`** (`control_interface.IControl`) – The control the frame will call to get access to the model.

`activate()`

Tells the current frame to activate and collect all the data it needs.

Returns

True, if activation was successful, false else

Return type

bool

src.osm_configurator.view.toplevelframes.project_head_frame module

class ProjectHeadFrame(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame shows the header pipeLine of the application, if a project is opened. Functionality the user can use: - Exit to the main menu - Save the project - Go to the settings - Change between different frames to edit configurations - Use exports

This frame is always on the top of the window. below it will be presented a frame to edit some part of the project and below that Frame will be a FootFrame. Exceptions are the MainMenu and the creation of a new project without this header.

__init__(*state_manager*, *control*)

This method creates a ProjectHeadFrame, letting the user navigate the pipeline and exit back to the main menu. The user can also open the settings, save the project or export the project.

Parameters

- **state_manager** (`state_manager.StateManager`) – The frame will call the StateManager, if it wants to switch states.
- **control** (`control_interface.IControl`) – The frame will call the control, to gain access to the model.

activate()

Tells the current frame to activate and collect all the data it needs.

Returns

True, if activation was successful, false else

Return type

bool

src.osm_configurator.view.toplevelframes.reduction_frame module

class ReductionFrame(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame lets the user edit the reduction of all the categories. It will consist of a list on the left to choose a category. On the right will be two sub-frames to change inbetween. On the right are two interchangeable sub-frames: One frame provides the configuration-options on how to calculate the Reduction. The other frame provides the default calculation-values.

__init__(*state_manager*, *control*)

This method creates a ReductionFrame, that lets the user edit the reduction of all the categories.

Parameters

- **state_manager** (`state_manager.StateManager`) – The frame will call the StateManager, if it wants to switch states.
- **control** (`control_interface.IControl`) – The control the frame will call to get access to the model.

activate()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.toplevelframes.settings_frame module

`class SettingsFrame`(*state_manager*, *control*)

Bases: *TopLevelFrame*

This frame shows the user the settings for: - The application - The current project

It either can be both or only the application settings.

`__init__`(*state_manager*, *control*)

This method creates a SettingsFrame, that lets the user set the application and project settings.

Parameters

- **state_manager** (`state_manager.StateManager`) – The StateManager the frame will call, if it wants to switch states.
- **control** (`control_interface.IControl`) – The control the frame will call to get access to the model.

`activate`()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.toplevelframes.top_level_frame module

`class TopLevelFrame`(*state_manager*, *control*)

Bases: ABC

This class describes a frame, that has a fully developed functionality and that can be placed on a window. A TopLevelFrame might have manageable frames below him.

`__init__`(*state_manager*, *control*)

This method defines how a TopLevelFrame is created.

Parameters

- **state_manager** (`state_manager.StateManager`) – The StateManager the frame will call, if it wants to switch states.
- **control** (`control_interface.IControl`) – The control the frame will call, to gain access to the model.

`abstract activate`()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

Module contents

src.osm_configurator.view.utilityframes package

Submodules

src.osm_configurator.view.utilityframes.export_frame module

class ExportFrame(*parent_frame*, *control*)

Bases: `object`

The ExportFrame provides a dropdown menu that providing the following Options: - project export - calculation export - Configurations Export

__init__(*parent_frame*, *control*)

This method creates an ExportFrame, that provides the user with different export options.

Parameters

- **parent_frame** (`project_head_frame.ProjectHeadFrame`) – The parent of the ExportFrame is the HeadFrame, where the export feature is located.
- **control** (`control_interface.IControl`) – The control the frame calls, to gain access to the model to export.

src.osm_configurator.view.utilityframes.reduction_calculation_frame module

class ReductionCalculationFrame(*parent*, *control*)

Bases: `object`

This frame provides the ability to the user to set how the calculation of the reduction of a category will be done. This is a subframe from the ReductionFrame.

__init__(*parent*, *control*)

This method creates a ReductionCalculationFrame, that lets the user edit the calculation of the reduction of Categories. :param parent: This is the parent frame of this frame. The frame will be located here. :type parent: reduction_frame.ReductionFrame :param control: The control the frame will call to get access to the model. :type control: control_interface.IControl

activate()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.utilityframes.reduction_default_value_frame module

class ReductionDefaultValueFrame(*parent*, *control*)

Bases: `object`

This frame shows a list of tags in a priority order, that can be expanded by adding or removing tags. These tags can hold default values on attributes, that can be used in the calculation.

__init__(*parent*, *control*)

This method creates a ReductionDefaultValueFrame where the User can edit default-values on tags for categories.

Parameters

- **parent** (`reduction_frame.ReductionFrame`) – This is the parent frame of this frame. The frame will be located here.
- **control** (`control_interface.IControl`) – The control the frame calls, to gain access to the model.

activate()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.utilityframes.settings_application_frame module

class SettingsApplicationFrame(*parent*, *control*)

Bases: `object`

This frame shows the settings of the application.

__init__(*parent*, *control*)

This method creates a SettingsApplicationFrame, showing the settings of the application.

Parameters

- **parent** (`settings_frame.SettingsFrame`) – The parent of this frame, where this frame will be located.
- **control** (`control_interface.IControl`) – The control the frame will call, to gain access to the Model and application settings.

activate()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.utilityframes.settings_project_frame module

class SettingsProjectFrame(*parent*, *control*)

Bases: `object`

This frame shows the current project settings.

__init__(*parent*, *control*)

This method creates a SettingsProjectFrame, showing the current project settings.

Parameters

- **parent** (`settings_frame.SettingsFrame`) – The parent of this frame, where this frame will be located.
- **control** (`control_interface.IControl`) – The control the frame will call, to gain access to the model.

activate()

Tells the current frame to activate and collect all the data it needs.

Returns
True, if activation was successful, false else

Return type
bool

src.osm_configurator.view.utilityframes.tag_list_frame module

class TagListFrame(*entries*)

Bases: `object`

This frame shows a textbox with a List of editable strings.

__init__(*entries*)

This method creates a textbox with the given entries.

Parameters
list[str] (*entries*) – A list of strings, that will be written in the textbox.

set_text_list(*entries*)

Replaces all shown textbox entries with the given text.

Parameters
list[str] (*entries*) – A list of strings, that will be shown on the textbox.

Returns
True if the replacement was successful, false else.

Return type
bool

get_text_list()

This method returns a list of strings containing the current textbox entries

Returns
List of strings containing the current textbox entries

Return type
list[str]

src.osm_configurator.view.utilityframes.tag_list_priority_frame module

`class TagListPriorityFrame`(*entries*)

Bases: `object`

This frame shows a list of tags (represented as strings). The tag-priority can be changed with arrows. The higher an entry is on the list, the lower its priority. A non-deletable DEFAULT-entry will always have the lowest priority.

`__init__`(*entries*)

This method will create a TagListPriorityFrame, showing a list of the given entries, ordered like the priorities.

Parameters

`list[str]` (*entries*) – List of strings, that shall be the entries on the priority list.

`set_tag_list`(*entries*)

Replaces all the entries with the new given entries, ordered in priority as the given List of strings is.

Parameters

`list[str]` (*entries*) – The list of strings, that shall be the entries on the priority list.

Returns

True, if the replacement was successful, false if not.

Return type

bool

`get_tag_list`()

Returns the current list of entries the priority list holds, ordered from lowest to highest.

Returns

The entry list of strings on the list, ordered from the lowest to highest priority.

Return type

list[str]

Module contents

Module contents

Module contents

Module contents

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX