pyapi_rts

Release 0.1

KIT-IAI-ESA

Feb 28, 2023

GENERAL

CHAPTER

ONE

GETTING STARTED

1.1 Installation

1.1.1 Installing Poetry

pyapi_rts uses Poetry to manage python dependencies and versions. Installation instructions for your operating system can be found here: Poetry.

After Poetry is installed, the necessary Python version and dependencies can be installed by running the `poetry install` command.

1.1.2 Generate classes from RSCAD components

Before you can use pyapi_rts, you need to generate the classes from the RSCAD components. These classes are not included in the pyapi_rts distribution.

1. Check the `pyapi_rts/class_extractor/COMPONENTS` directory. If it exisists and is not empty, you can skip this step. Otherwise, copy the content of the COMPONENTS directory from the RSCAD distribution to the `pyapi_rts/class_extractor/COMPONENTS` directory. On Windows, this directory likely can be found at `C:\Program Files\RTDS\RSCAD FX x.x\MLIB\COMPONENTS`
2. Run `poetry run python ./pyapi_rts/class_extractor/main.py`. For options and more information, see *Class Extractor Usage*.

1.1.3 Check for errors

It is recommended to run the unit tests after executing the **ClassExtractor** to ensure no errors occured. To do this, run `poetry run pytest`.

1.2 Examples

See *Examples* for examples of API usage.

1.3 Development

1.3.1 Setting up a development environment

1. Update the dependencies: `poetry update`
2. Run `poetry install`.
3. To open a shell within the virtual environment of the project, run `poetry shell`.
4. Run ClassExtractor.

When using Visual Studio Code, the following extensions are recommended:

- autodocstring
- Coverage Gutters
- Python
- reStructuredText
- Jupyter Notebooks

1.3.2 Testing

- **Tests: `poetry run pytest`** Tests use the Python unittest framework.
- **Coverage:**

```
poetry run coverage run --omit */docs/*,*/tests/*,*/generated/*,.eggs/*,*/hooks/
↪* -m pytest
poetry run coverage report
poetry run coverage xml
```

1.3.3 Generating documentation

The documentation is created using Sphinx, which is a Python documentation generator. It uses restructured text (reST) as its markup language, a language similar to markdown. All relevant files for the documentation are located in the `docs` directory.

To generate the documentation, switch to the `docs` directory and run `make html` or `make latex`. After changes to the API, you can manually delete the `docs/apidoc` directory and regenerate it by running `poetry run sphinx-apidoc ./pyapi_rts/ */tests/* */generated/* -o ./docs/apidoc` from the pyapi_rts directory.

It is not recommended to do this, as the documentation is generated automatically by the pipeline on the main git branch.

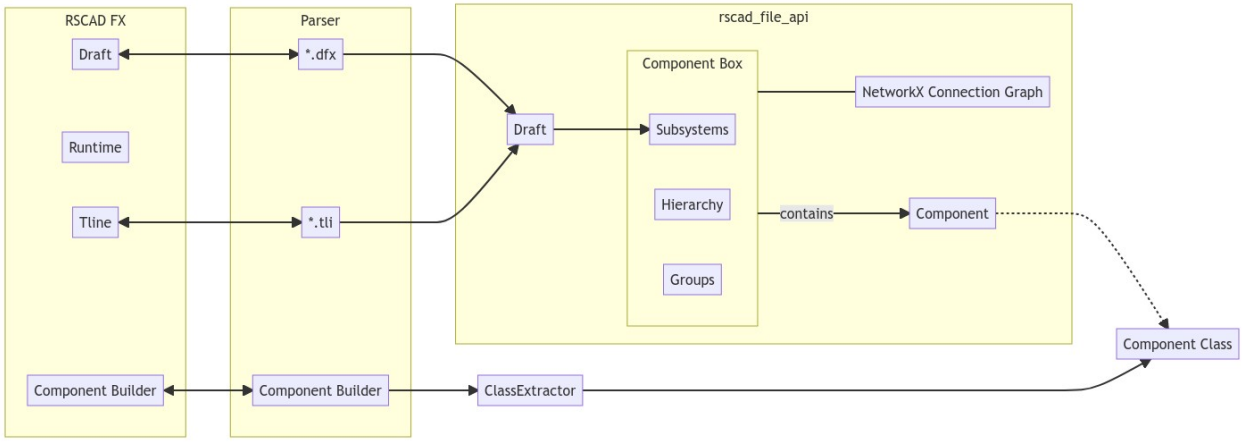
CHAPTER

TWO

OVERVIEW

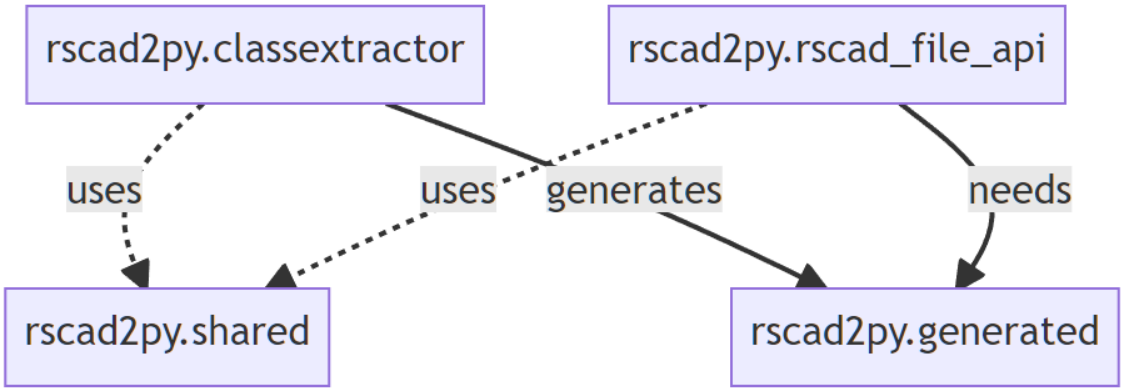
This page gives an overview of the structure of the project.

2.1 Structure



2.2 Modules

The project is divided into several modules.



The package meant to be used by the user is the `pyapi_rts.api` module. It contains the classes and functions that are used to read and write RSCAD files and edit the models.

The `pyapi_rts.class_extractor` module contains everything needed to extract the classes from the Component Builder files and extensions and hooks. It has no dependencies on the `pyapi_rts.api` module. The module is used to store all of the data necessary to generate the classes from the Component Builder files. It is not needed for the distribution of projects built using the API. The results of the process are stored in the `pyapi_rts.generated` module.

The `pyapi_rts.shared` module contains the classes that are shared between the api and class_extractor modules.

2.3 Development

The development is strongly affected by the 'black box' of the RSCAD FX program. This requires a lot of manual work figuring out the behavior of RSCAD and its included tools.

This problem becomes even more severe when the RSCAD FX program is updated or new features are added to pyapi_rts. For that reason, the development of pyapi_rts heavily relies on testing and modularization into mostly self-contained modules and features that are easy to test.

Ideally, the tests are written first and checked in with the code. Additionally, assumptions about the behavior of RSCAD FX should be documented and represented by a known good RSCAD model.

CHAPTER

THREE

CONNECTION GRAPH

3.1 Introduction

The **Connection Graph** is generated for each *Component Box* in the model. It represents the connection between the components using the components as nodes.

Using an additional **Link Dictionary**, the connection graphs themselves can be merged into a graph of the whole model or just identify the connections to other *Component Boxes*.

3.2 When are two components connected?

Whether two components are connected differs between the graph itself , the get_connected_to() method and the get_connected_at_point() method.

Type	Advantages	Disadvantages	Rules	Hook available
Graph	- Internal - Accurate to RSCAD draft mode - Lazily evaluated and cached	- Not across Component Boxes - Only UUIDs are returned	At least connection point of the two components overlap.	Yes
get_connected_to()	- Easy to use - Uses (cached) graph when available - Mostly accurate to RSCAD simulation	- Inflexible	Connected on graph or by name	No
get_connected_at_point()	- Useful for following signal from specific connection point - Simple search for i.e. manager	- Doesn't use caching	Connected between the connection points only via 'connecting components', i.e. bus etc.	No

Most of the time, only one of these options is suited for a specific use case.

3.3 Generation and updates

The connection graph is generated on first use to avoid long delays when the model is loaded. It is updated whenever a component is added, removed or modified in a way that changes the connection points.

Every time the graph is generated or updated, the **Position Dictionary** and **Link Dictionary** are also updated.

3.4 Cloned and referenced components

To improve performance, the `get_components()` method in the *ComponentBox* class has a *clone* parameter. If this is set to True, the returned components are clones of the original components. If this is set to False, the returned components are references to the original components.

Changing referenced components can cause the connection graph to be invalid. This can not be detected by pyapi_rts and should be avoided.

CHAPTER

FOUR

TYPES OF CONNECTIONS

- **grid-based connections**
  - e.g. via BUS or WIRE
- **grid-based connections over multiple hierarchies**
  - e.g. BUSLABEL connected to HIERARCHY via BUS
  - characterized by node type NAME_CONNECTED (?)
- **label based connections**
  - i.e. wirelabel and signal names of components
- **linked node connections**
  - connect over one or multiple hierarchies without grid connection
  - characterized by node type NAME_CONNECTED:LINKED
  - used in rtds_sharc_node and rtds_sharc_sld_BUSLABEL
- **cross-rack connections**
  - line, cable and cross-rack transformer
  - signal import/export

CHAPTER

FIVE

GLOSSARY

5.1 General

RSCAD FX Software developed by RTDS for the configuration, execution and analysis of real-time simulations. Link.

CBuilder Software for creation and editing of **Component Builder files**. Included in RSCAD FX distributions.

Runtime Software for execution and monitoring of real-time simulations on RTDS hardware simulators. Included in RSCAD FX distributions.

TLine Software for creation and editing of **TLine (*.tli) files**. Included in RSCAD FX distributions.

5.2 pyapi_rts

Draft The information from a *.dfx file. Can contain one or more **subsystems** and some metadata.

Subsystem A canvas on which **components** and **component boxes** are placed.

Component Box A set of components on a canvas and their connections to each other. The basis for **Subsystems**, **Hierarchies** and **Groups**. See *Connection Graph* for more information.

Hierarchy A component that is a **Component Box** at the same time. Through connections to the component, connections to components within the **Component Box** can be established. A hierarchy can contain other hierarchies, allowing for better readability of the model.

Group Technically a **component**, but the contained components are drawn on the parent canvas. In RSCAD FX, grouped components can be moved together and not edited while grouped.

Component A element that can be placed on a canvas. A component has a type, which are defined in *Component Builder files*.

5.3 File Types

Component Builder A Component Builder file describes how a given component is drawn on the canvas, what connections and parameters it has and more.

***.dfx: Draft** Contains the draft of a model, meaning the subsystems and metadata.

***.tli: TLine** Describes the properties of a type of transmission line.

CHAPTER

SIX

PYAPI_RTS

6.1 pyapi_rts package

6.1.1 Subpackages

pyapi_rts.api package

Subpackages

pyapi_rts.api.lark package

Submodules

pyapi_rts.api.lark.rlc_tline module

class pyapi_rts.api.lark.rlc_tline.**RLCTLine**(*name: str*, *tli_file: Optional[*pyapi_rts.api.lark.tli_transformer.TliFile*] = None*)

Bases: `object`

A TliFile wrapper that simplifies data entry for metric RLC Options in ohms.

property frequency: float

Steady State Frequency

Returns Steady State Frequency in Hz

Return type float

classmethod from_file(*file_path: str*) → *pyapi_rts.api.lark.rlc_tline.RLCTLine*

Creates an RLCTline from a file. Raises ValueError if the file contains the wrong data.

Parameters **file_path** (*str*) – The path to the file.

Returns The RLCTline created from the file.

Return type RLCTline

property ground_resistivity: float

Ground Resistivity

Returns Ground Resistivity in Ohm*m

Return type float

property length: float

Line Length

Returns Line Length in km

Return type float

property mutual_reactance: float

Mutual Resistance

Returns Mutual Reactance in Ohm/km

Return type float

property mutual_resistance: float

Mutual Resistance

Returns Mutual Resistance in Ohm/km

Return type float

property num_phases: int

Number of Phases

Returns Number of Phases

Return type int

property r0: float

Zero Sequence Series Resistance

Returns Zero Sequence Series Resistance in Ohm/km

Return type float

property r1: float

Positive Sequence Series Resistance

Returns Positive Sequence Series Resistance in Ohm/km

Return type float

property transposed: bool

Ideally Transposed

Returns True if lines are ideally transposed

Return type bool

write_file(*directory: str*) → bool

Writes the .tli file to the directory. Uses the object's name as file name.

Parameters **directory** (*str*) – The directory to write the .tli file to.

Returns True if the file was written, False otherwise.

Return type bool

property xcap0: float

Zero Sequence Series Cap Reactance

Returns Zero Sequence Series Cap Reactance in MOhm*km

Return type float

property xcap1: float

Positive Sequence Series Cap Reactance

Returns Positive Sequence Series Cap Reactance in MOhm*km

Return type float

property xind0: float

Zero Sequence Series Ind Reactance

Returns Zero Sequence Series Ind Reactance in Ohm/km

Return type float

property xind1: float

Positive Sequence Series Ind Reactance

Returns Positive Sequence Series Ind Reactance in Ohm/km

Return type float

pyapi_rts.api.lark.tli_transformer module

class pyapi_rts.api.lark.tli_transformer.**TliDataType**(*value*)

Bases: enum.Enum

Enum for the different data types in the tli file.

ANY = 1

Allow both data types, keys in upper case use metadata, keys in lower case use data if both exist at path.

DATA = 2

Datatype from key-value entries in TliSections.

METADATA = 3

Metadata, defined in TliRtdsMetadata.

SECTION = 4

Section, defined in TliSections.

class pyapi_rts.api.lark.tli_transformer.**TliFile**

Bases: object

The class for a .tli file.

classmethod from_file(*file_path: str*) → *pyapi_rts.api.lark.tli_transformer.TliFile*

Creates a TliFile from a file.

Parameters **file_path** (*str*) – The path to the file.

Returns The TliFile created from the file.

Return type *TliFile*

get(*path: str*, *data_type:* pyapi_rts.api.lark.tli_transformer.TliDataType = *TliDataType.ANY*) → str | *pyapi_rts.api.lark.tli_transformer.TliSection*

Gets the value of the key at the path.

Parameters

- **path** (*str*) – The path to the key through the sections, split by '/'.

- **data_type** (TliDataType) – The type of data to search for at path.

Returns The value of the key at the path.

Return type str

read_file(*path: str*) → None

Reads a .tli file from the path and fills the object with the data :param path: The path to the .tli file :type path: str :return: None

write_file(*path: str*) → bool

Writes the .tli file to the path.

Parameters **path** (*str*) – The path to write the .tli file to.

Returns True if the file was written, False otherwise.

Return type bool

class pyapi_rts.api.lark.tli_transformer.**TliRtdsMetadata**(*key*, *value*)

Bases: object

Contains key-value metadata in *.tli files

key

The key of the metadata

value

The value of the metadata

class pyapi_rts.api.lark.tli_transformer.**TliSection**(*title*, *value=None*)

Bases: object

Contains a section in a *.tli file. The title can be a string or key-value pair.

dictionary

The key-value pairs contained in this section

get(*path: str*, *data_type:* pyapi_rts.api.lark.tli_transformer.TliDataType = *TliDataType.ANY*) → str

Returns the data, metadata or section at the given path.

Parameters

- **path** (*str*) – Path to the section. If it only contains whitespace, returns the section itself.
- **data_type** (TliDataType) – The type of data to search for at path.

Returns The section at the given path

Return type *TliSection*

metadata: list[*pyapi_rts.api.lark.tli_transformer.TliRtdsMetadata*]

The key-value pairs starting with '!RTDS_' in *.tli files.

sections: list[*pyapi_rts.api.lark.tli_transformer.TliSection*]

The sections contained in this section

title_key: str

The title of the section.

title_value: str | None

The value of the title if it is a key-value pair. None otherwise.

write() → str

Returns the section as a string.

class pyapi_rts.api.lark.tli_transformer.**TliTransformer**

Bases: lark.visitors.Transformer

Transformer for the lark parser for .tli files

dict(*items*)

pair(*args*)

rtds_meta(*content*)

section(*items*)

start(*items*)

value(*val*)

Module contents

Contains the lark parsers and transformers for some RSCAD file types.

class pyapi_rts.api.lark.**TliTransformer**

Bases: lark.visitors.Transformer

Transformer for the lark parser for .tli files

dict(*items*)

pair(*args*)

rtds_meta(*content*)

section(*items*)

start(*items*)

value(*val*)

pyapi_rts.api.parameters package

Submodules

pyapi_rts.api.parameters.boolean_parameter module

class pyapi_rts.api.parameters.boolean_parameter.**BooleanParameter**(*key: str*, *value: bool*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A boolean parameter

get_value() → bool

Get the value of the parameter

Returns The value of the parameter

Return type bool

set_str(*value: str*) → bool

Set the value of the parameter from a string

Parameters **value** (*str*) – The value to set

Returns Success of the operation

Return type bool

set_value(*value: bool*) → bool

Set the value of the parameter

Parameters **value** (*bool*) – The value to set

Returns Success of the operation

Return type bool

pyapi_rts.api.parameters.color_parameter module

class pyapi_rts.api.parameters.color_parameter.**ColorParameter**(*key*, *value*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.string_parameter.StringParameter*

default: Any = '#000000'

Default value for the parameter

pyapi_rts.api.parameters.connection_point module

class pyapi_rts.api.parameters.connection_point.**ConnectionPoint**(*x: int | str*, *y: int | str*, *name: str*, *io: pyapi_rts.shared.node_type.NodeIO*, *component*, *link: tuple[pyapi_rts.shared.node_type.NodeType, str] = (<NodeType.OTHER: 'OTHER'>, '')*)

Bases: object

A connection point of a component rectangle.

component: *Component*

The component this connection point belongs to.

io

IO Type of the connection point.

link: str

Link name.

property link_name: str

The link name or the name of the connection point if no link is defined. :return: The key for the link dictionary. :rtype: str

link_type: *NodeType*

Linking behaviour to other nodes.

name

Name of the connection point.

property position: tuple[int, int]

property position_abs: tuple[int, int]

position_from_dict(*comp_dict: dict*, *absolute: bool = False*) → tuple[int, int]

x: *ParameterBoundProperty*

X position relative to the center of the component.

y

Y position relative to the center of the component.

pyapi_rts.api.parameters.float_parameter module

class pyapi_rts.api.parameters.float_parameter.**FloatParameter**(*key: str*, *value: float*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A parameter containing a floating point number.

default: Any = 0.0

Default value for the parameter

get_value() → float

Returns the value of the parameter.

Returns The value of the parameter

Return type float

get_value_as_int() → int

Get the value of the parameter as an integer

Returns The value of the parameter as an integer

Return type int

set_str(*value: str*) → bool

Sets the value of the parameter from a string.

Parameters **value** (*str*) – The value of the parameter as a string

Returns Success of the operation

Return type bool

set_value(*value: float*) → bool

Sets the value of the parameter.

Parameters **value** (*float*) – The value of the parameter

Returns Success of the operation

Return type bool

pyapi_rts.api.parameters.integer_parameter module

class pyapi_rts.api.parameters.integer_parameter.**IntegerParameter**(*key: str*, *value: int*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A parameter containing an integer number.

default: Any = 0

Default value for the parameter

get_value() → int

Returns the value of the parameter.

Returns The value of the parameter

Return type int

get_value_as_int() → int

Returns the value of the parameter as an integer.

Returns The value of the parameter.

Return type int

set_str(*value: str*) → bool

Sets the value of the parameter from a string.

Parameters **value** (*str*) – The value of the parameter as a string

Returns Success of the operation

Return type bool

set_value(*value: int*) → bool

Sets the value of the parameter.

Parameters **value** (*int*) – The value of the parameter

Returns Success of the operation

Return type bool

pyapi_rts.api.parameters.name_parameter module

class pyapi_rts.api.parameters.name_parameter.**NameParameter**(*key: str*, *value: str*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A parameter containing a string representing a name.

default: Any = ''

Default value for the parameter

get_value() → str

Returns the value of the parameter.

Returns The value of the parameter

Return type str

get_value_as_int() → str

Returns the value of the parameter as an integer.

Returns The value of the parameter.

Return type str

set_str(*value: str*) → bool

Sets the value of the parameter from a string.

Parameters **value** (*str*) – The value of the parameter as a string

Returns Success of the operation

Return type bool

set_value(*value: str*) → bool

Sets the value of the parameter.

Parameters **value** (*str*) – The value of the parameter

Returns Success of the operation

Return type bool

pyapi_rts.api.parameters.parameter module

class pyapi_rts.api.parameters.parameter.**Parameter**(*key: str*, *value: Any*, *from_str: bool = False*)

Bases: object

Base class for all parameters

default: Any = None

Default value for the parameter

get_value() → Any

Get the value of the parameter

Returns The value of the parameter

Return type Any

get_value_as_int() → int

Get the value of the parameter as an integer

Returns The value of the parameter as an integer

Return type int

key: str

The key of the parameter

set_str(*value: str*) → bool

Set the value of the parameter from a string

Parameters **value** (*str*) – The value to set

Returns Success of the operation

Return type bool

set_value(*value: Any*) → bool

Set the value of the parameter

Parameters **value** (*Any*) – The value to set

Returns Success of the operation

Return type bool

pyapi_rts.api.parameters.parameter_collection module

class pyapi_rts.api.parameters.parameter_collection.**ParameterCollection**

Bases: object

A collection of specific parameters with certain keys and types

as_dict() → dict[str, *pyapi_rts.api.parameters.parameter.Parameter*]

get_value(*key: str*) → Optional[Any]

Returns the value of the parameter with the given key.

has_key(*key: str*) → bool

Checks if any parameter in collection has a given key

Parameters **key** (*str*) – The key to check for

Returns True if any parameter in collection has a given key

Return type bool

set_str(*key: str*, *value: str*) → bool

Tries to set parameter with given key to a value

Parameters

- **key** (*str*) – The key of the parameter to set
- **value** (*str*) – The string representation of the value to set the parameter to

Returns True if parameter was set, False if not

Return type bool

set_value(*key: str*, *value: Any*) → bool

Tries to set parameter with given key to a value

Parameters

- **key** (*str*) – The key of the parameter to set
- **value** (*Any*) – The value to set the parameter to

Returns True if parameter was set, False if not

Return type bool

pyapi_rts.api.parameters.string_parameter module

class pyapi_rts.api.parameters.string_parameter.**StringParameter**(*key: str*, *value: str*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A parameter that contains a string

default: Any = ''

Default value for the parameter

get_value() → str

Returns the value of the parameter

Returns The value of the parameter

Return type str

get_value_as_int() → int

Returns the value of the parameter as an integer.

Returns The value of the parameter.

Return type int

set_str(*value: str*) → bool

Sets the value of the parameter to the given string

Parameters **value** (*str*) – The value to set

Returns True if the value was set, False otherwise

Return type bool

set_value(*value: str*) → bool

Sets the value of the parameter

Parameters **value** (*str*) – The value to set

Returns True if the value was set, False otherwise

Return type bool

Module contents

Classes for handling parameters in RSCAD files.

class pyapi_rts.api.parameters.**BooleanParameter**(*key: str*, *value: bool*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A boolean parameter

get_value() → bool

Get the value of the parameter

Returns The value of the parameter

Return type bool

set_str(*value: str*) → bool

Set the value of the parameter from a string

Parameters **value** (*str*) – The value to set

Returns Success of the operation

Return type bool

set_value(*value: bool*) → bool

Set the value of the parameter

Parameters **value** (*bool*) – The value to set

Returns Success of the operation

Return type bool

class pyapi_rts.api.parameters.**ColorParameter**(*key*, *value*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.string_parameter.StringParameter*

default: Any = '#000000'

Default value for the parameter

key: str

The key of the parameter

class pyapi_rts.api.parameters.**ConnectionPoint**(*x: int | str*, *y: int | str*, *name: str*, *io: pyapi_rts.shared.node_type.NodeIO*, *component*, *link: tuple[pyapi_rts.shared.node_type.NodeType, str] = (<NodeType.OTHER: 'OTHER'>, '')*)

Bases: object

A connection point of a component rectangle.

component: *Component*

The component this connection point belongs to.

io

IO Type of the connection point.

link: str

Link name.

property link_name: str

The link name or the name of the connection point if no link is defined. :return: The key for the link dictionary. :rtype: str

link_type: *NodeType*

Linking behaviour to other nodes.

name

Name of the connection point.

property position: tuple[int, int]

property position_abs: tuple[int, int]

position_from_dict(*comp_dict: dict*, *absolute: bool = False*) → tuple[int, int]

x: *ParameterBoundProperty*

X position relative to the center of the component.

y

Y position relative to the center of the component.

class pyapi_rts.api.parameters.**FloatParameter**(*key: str*, *value: float*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A parameter containing a floating point number.

default: Any = 0.0

Default value for the parameter

get_value() → float

Returns the value of the parameter.

Returns The value of the parameter

Return type float

get_value_as_int() → int

Get the value of the parameter as an integer

Returns The value of the parameter as an integer

Return type int

key: str

The key of the parameter

set_str(*value: str*) → bool

Sets the value of the parameter from a string.

Parameters **value** (*str*) – The value of the parameter as a string

Returns Success of the operation

Return type bool

set_value(*value: float*) → bool

Sets the value of the parameter.

Parameters **value** (*float*) – The value of the parameter

Returns Success of the operation

Return type bool

class pyapi_rts.api.parameters.**IntegerParameter**(*key: str*, *value: int*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A parameter containing an integer number.

default: Any = 0

Default value for the parameter

get_value() → int

Returns the value of the parameter.

Returns The value of the parameter

Return type int

get_value_as_int() → int

Returns the value of the parameter as an integer.

Returns The value of the parameter.

Return type int

key: str

The key of the parameter

set_str(*value: str*) → bool

Sets the value of the parameter from a string.

Parameters **value** (*str*) – The value of the parameter as a string

Returns Success of the operation

Return type bool

set_value(*value: int*) → bool

Sets the value of the parameter.

Parameters **value** (*int*) – The value of the parameter

Returns Success of the operation

Return type bool

class pyapi_rts.api.parameters.**NameParameter**(*key: str*, *value: str*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A parameter containing a string representing a name.

default: Any = ''

Default value for the parameter

get_value() → str

Returns the value of the parameter.

Returns The value of the parameter

Return type str

get_value_as_int() → str

Returns the value of the parameter as an integer.

Returns The value of the parameter.

Return type str

key: str

The key of the parameter

set_str(*value: str*) → bool

Sets the value of the parameter from a string.

Parameters **value** (*str*) – The value of the parameter as a string

Returns Success of the operation

Return type bool

set_value(*value: str*) → bool

Sets the value of the parameter.

Parameters **value** (*str*) – The value of the parameter

Returns Success of the operation

Return type bool

class pyapi_rts.api.parameters.**Parameter**(*key: str*, *value: Any*, *from_str: bool = False*)

Bases: object

Base class for all parameters

default: Any = None

Default value for the parameter

get_value() → Any

Get the value of the parameter

Returns The value of the parameter

Return type Any

get_value_as_int() → int

Get the value of the parameter as an integer

Returns The value of the parameter as an integer

Return type int

key: str

The key of the parameter

set_str(*value: str*) → bool

Set the value of the parameter from a string

Parameters **value** (*str*) – The value to set

Returns Success of the operation

Return type bool

set_value(*value: Any*) → bool

Set the value of the parameter

Parameters **value** (*Any*) – The value to set

Returns Success of the operation

Return type bool

class pyapi_rts.api.parameters.**ParameterCollection**

Bases: object

A collection of specific parameters with certain keys and types

as_dict() → dict[str, *pyapi_rts.api.parameters.parameter.Parameter*]

get_value(*key: str*) → Optional[Any]

Returns the value of the parameter with the given key.

has_key(*key: str*) → bool

Checks if any parameter in collection has a given key

Parameters **key** (*str*) – The key to check for

Returns True if any parameter in collection has a given key

Return type bool

set_str(*key: str*, *value: str*) → bool

Tries to set parameter with given key to a value

Parameters

- **key** (*str*) – The key of the parameter to set
- **value** (*str*) – The string representation of the value to set the parameter to

Returns True if parameter was set, False if not

Return type bool

set_value(*key: str*, *value: Any*) → bool

Tries to set parameter with given key to a value

Parameters

- **key** (*str*) – The key of the parameter to set
- **value** (*Any*) – The value to set the parameter to

Returns True if parameter was set, False if not

Return type bool

class pyapi_rts.api.parameters.**StringParameter**(*key: str*, *value: str*, *from_str: bool = False*)

Bases: *pyapi_rts.api.parameters.parameter.Parameter*

A parameter that contains a string

default: Any = ''

Default value for the parameter

get_value() → str

Returns the value of the parameter

Returns The value of the parameter

Return type str

get_value_as_int() → int

Returns the value of the parameter as an integer.

Returns The value of the parameter.

Return type int

key: str

The key of the parameter

set_str(*value: str*) → bool

Sets the value of the parameter to the given string

Parameters **value** (*str*) – The value to set

Returns True if the value was set, False otherwise

Return type bool

set_value(*value: str*) → bool

Sets the value of the parameter

Parameters **value** (*str*) – The value to set

Returns True if the value was set, False otherwise

Return type bool

Submodules

pyapi_rts.api.component module

class pyapi_rts.api.component.**Component**(*type_name: Optional[str] = None*, *stretchable:* pyapi_rts.shared.stretchable.Stretchable *= Stretchable.NO*, *linked: Optional[bool] = None*)

Bases: pyapi_rts.api.internals.dfxblock.DfxBlock

A RSCAD component

GRID_SIZE = 32

LOAD_UNITS_DEFAULT = 10

LOAD_UNIT_NAMES = ['loadunit', 'LoadUnit', 'load']

abstract as_dict() → dict[str, *pyapi_rts.api.parameters.parameter.Parameter*]

Returns the parameters of the component as a dictionary

Returns The parameters of the component as a dictionary

Return type dict[str, *Parameter*]

block() → list[str]

Returns the component as a .dfx block

Returns The component as a .dfx block

Return type list[str]

property bounding_box: tuple[int, int, int, int]

property bounding_box_abs: tuple[int, int, int, int]

bounding_box_from_dict(*dictionary: dict*, *absolute: bool = False*) → tuple[int, int, int, int]

property connection_points: dict[str, *pyapi_rts.api.parameters.connection_point.ConnectionPoint*]

connection_points_from_dict(*dictionary*) → dict[str, *pyapi_rts.api.parameters.connection_point.ConnectionPoint*]

duplicate() → *pyapi_rts.api.component.Component*

Creates a copy of the component with the same UUID

Returns The copy of the component

Return type *Component*

generate_pos_dict() → dict[str, list[tuple]]

Creates a dictionary that maps positions to connection points Key: "{x-coord},{y-coord}" of connection point Value: tuple of name of connection point and id of component

Returns The created dictionary

Return type dict[str, list[tuple]]

get_by_key(*key: str*, *default_value: Optional[Any] = None*, *as_int: bool = False*) → Optional[Any]

Returns the parameter with a certain key

Parameters

- **key** (*str*) – The key of the parameter
- **default_value** (*Any, optional*) – The default value if the parameter is not found, defaults to None

Returns The parameter or the default value if not found

Return type Any | None

get_connected_at_point(*point_name: str*, *return_connecting: bool = False*, *component_type: Optional[str] = None*) → list['Component']

Returns a list of all components connected at the connection point with the given name. Filters for components of a given type if component_type is specified.

Parameters

- **point_name** (*str*) – Name of the connection point
- **return_connecting** (*bool, optional*) – Returns the connecting components if True, otherwise only the end components are returned, defaults to False
- **component_type** (*str, optional*) – The type of components to return, defaults to None

Returns list of connected components

Return type list[*Component*]

get_special_value(*key: str*) → str

Returns the special value of the component. :param key: The key of the special value. :type key: str :return: The special value or empty string if not found. :rtype: str

graph_similar_to(*comp:* pyapi_rts.api.component.Component) → bool

Checks if the two components are identical for the purposes of the graph reperesentation. That is the case if: 1. They have the same id 2. They have the same coordinates, mirror and rotation 3. They have the same rectangle position and size

Parameters **comp** (Component) – The component to check for similarity.

Returns True if the two components are identical (for the graph).

Return type bool

has_key(*key: str*) → bool

Checks if a parameter with a certain key exists

Parameters **key** (*str*) – The key of the parameter

Returns True if the parameter exists, False otherwise

Return type bool

property height: int

property is_connecting: bool

Whether the component is a connecting component like wire or bus

Returns Whether the component is connecting

Return type bool

property is_hierarchy_connecting: bool

Whether the component is a connecting hierarchies without being a component box.

Returns Whether the component is hierarchy connecting

Return type bool

property is_label: bool

Whether the component is a label

Returns Whether the component is a label

Return type bool

property load_units: int

Returns the load units of the component based on the data available. :return: The load units of the component. :rtype: int

property mirror: int

The mirror state of the component

Returns The mirror of the component (0: no mirror, 1: mirror)

Return type int

property name: str

The parameter with key 'Name' with the enumerator applied

overlaps(*other:* pyapi_rts.api.component.Component) → bool

Checks if the rectangles overlap.

Parameters **other** (Component) – Another component

Returns True if the rectangles overlap, False otherwise

Return type bool

parent: *ComponentBox*

The component that contains this component.

read_block(*block: pyapi_rts.api.internals.block.Block*, *check=True*)

Reads a component from a list of lines

Parameters

- **block** (*Block*) – A list of lines describing the component

- **check** (*bool, optional*) – Checks the block format before parsing, defaults to True

property rotation: int

The rotation of the component

Returns The rotation of the component (times 90 degrees)

Return type int

set_by_key(*key: str*, *value: Any*) → bool

Sets a parameter with a certain key

Parameters

- **key** (*str*) – The key of the parameter
- **value** (*Any*) – The value of the parameter

Returns True if the parameter was set successfully, False otherwise

Return type bool

stretchable: *Stretchable*

Stretchable dimensions of the component

touches(*comp:* pyapi_rts.api.component.Component) → list[tuple[*pyapi_rts.api.parameters.connection_point.ConnectionPoint*, *pyapi_rts.api.parameters.connection_point.ConnectionPoint*]]

Returns a list of connection points the two components touch at the same time

Parameters **comp** (Component) – The component to check for touching connection points

Returns A list of connection points the two components touch at the same time

Return type list[tuple[*ConnectionPoint*, *ConnectionPoint*]]

property type: str

The component type

Returns The component type

Return type str

property uuid: str

Returns the component uuid

Returns The component UUID

Return type str

property width: int

property x: int

The x coordinate of the component

Returns The x coordinate of the component

Return type int

property x1: int

property x2: int

property y: int

The y coordinate of the component

Returns The y coordinate of the component

Return type int

property y1: int

property y2: int

pyapi_rts.api.component_box module

class pyapi_rts.api.component_box.**ComponentBox**(*parent=None*)

Bases: object

Abstract class for an object containing a list of components

add_component(*component:* pyapi_rts.api.component.Component) → None

Add a component to the component box and update the connection graph and other data structures.

Parameters **component** (Component) – The component to add to this box

box_parent

The parent component box of this component box

generate_full_graph() → tuple[networkx.classes.graph.Graph, dict]

Generate the full graph consisting of the union of all componentBoxes included in this one.

Returns The graph and dictionary of cross-hierarchy connection points.

Return type tuple[Graph, dict]

get_at_point(*uuid: str*, *point_name: str*) → list[tuple[str, str]]

Returns a list of connection points at a given position on the grid.

Parameters

- **uuid** (*str*) – The UUID of the component to search from.
- **point_name** (*str*) – The name of the connection point to search from.

Returns list of (uuid, point_name) tuples.

Return type list[tuple[str, str]]

get_box_type() → int

Returns the type of the component box. :return: The type of the component box. :rtype: int

get_by_id(*cid: str*, *recursive: bool = True*, *with_groups=True*) → *pyapi_rts.api.component.Component* | None

Get a component by its id

Parameters

- **cid** (*str*) – Component UUID to search for
- **recursive** (*bool, optional*) – Searches recursively in boxes, defaults to True
- **with_groups** (*bool, optional*) – Include components in groups, defaults to True

Returns Component with the given UUID if found, None otherwise

Return type *Component* | None

get_component_boxes(*recursive: bool = False*) → list['ComponentBox']

Returns a list of all component boxes in the component box.

get_components(*recursive=False*, *clone=True*, *with_groups=False*) → list[*pyapi_rts.api.component.Component*]

Returns a list of all components in the component box.

Parameters

- **recursive** (*bool, optional*) – Also lists components in component boxes contained in this, defaults to False.
- **copy** (*bool, optional*) – Returns a copy of the list instead of the list itself, defaults to True
- **with_groups** (*bool, optional*) – Include components in groups, defaults to False

Returns list of components in the component box

Return type list[*Component*]

get_connected_at_component_point(*uuid: str*, *point_name: str*, *return_connecting: bool = False*, *component_type: Optional[str] = None*, *callers: list['ComponentBox'] = []*) → list[*pyapi_rts.api.component.Component*]

Returns a list of all components connected at the connection point with the given name. Filters for components of a given type if component_type is specified.

Parameters

- **point_name** (*str*) – Name of the connection point
- **component_type** (*str or None optional*) – Only return components of this type, defaults to None
- **callers** (*list[ComponentBox] optional*) – list of components that have already been called, defaults to []

Returns list of all components connected to the given label

Return type list[*Component*]

get_connected_to(*component:* pyapi_rts.api.component.Component, *clone: bool = True*, *include_all_connections: bool = False*) → list[*pyapi_rts.api.component.Component*]

Returns all components connected to a certain component, including those from hierarchies

Parameters

- **component** (Component) – Initial component to search from
- **clone** (*bool, optional*) – Whether to clone the components, defaults to True
- **include_all_connections** (*bool, optional*) – Whether to include non-signal connections, e.g. TLINE to calc block.

Returns list of all components connected to the given component

Return type list[*Component*]

get_connected_to_label(*label_name: str*, *return_connecting: bool = False*, *callers=[]*) → list[*pyapi_rts.api.component.Component*]

Returns all components connected to a wire or bus with a label with the given name. Returns the empty list if the label does not exist.

Parameters

- **label_name** (*str*) – The label of the bus or wire connection
- **return_connecting** (*bool*) – If true, returns the connecting components.
- **callers** (*list[ComponentBox]*) – list of ComponentBoxes that have already been called.

Returns list of all components connected to the given label

Return type list[*Component*]

get_connection_graph() → networkx.classes.graph.Graph

Returns the connection graph and generates it if it is not already generated.

The connection graph only contains connections in the same hierarchy level and does not include connections via wire label. This method also triggers the generation of the link dictionary.

Returns The connection graph

Return type Graph

get_draft()

Returns the draft of the component box.

Returns The draft this component box is part of

Return type *pyapi_rts.api.draft.Draft*

get_groups() → list['ComponentBox']

Returns a list of all groups in the component box.

Returns list of groups in the component box

Return type list[*Group*]

get_hierarchies(*recursive=False*) → list[*pyapi_rts.api.component.Component*]

Returns all hierarchy components in the component box

Parameters **recursive** (*bool, optional*) – Recusive search, defaults to False

Returns list of all hierarchies in the component box

Return type list[*Component*]

get_link_dict() → dict[str, list[tuple[str, str, *pyapi_rts.shared.node_type.NodeType*]]]

Returns the link dictionary and generates it if it is not already generated.

The link dictionary links the name of a connection point to a list of component UUIDs. It only includes NAME_CONNECTED connection points, e.g. of bus labels and wire labels.

Returns The link dictionary; (Component.uuid, ConnectionPoint.name, ConnectionPoint.link_type)

Return type dict[str, list[tuple[str, str, *NodeType*]]]

get_rack_type() → int

Returns the rack type of the component box. :return: The rack type of the component box. :rtype: int

modify_component(*component:* pyapi_rts.api.component.Component, *recursive=True*) → bool

Modify a component in the component box and update the connection graph and other data structures.

Parameters

- **component** (Component) – The component to modify
- **recursive** (*bool, optional*) – Searches recursively, defaults to True

Returns Success of search and modification

Return type bool

remove_component(*cid: str*, *recursive: bool = False*, *with_groups=True*) → bool

Remove a component from the component box and update the connection graph and other data structures.

Parameters

- **cid** (*str*) – Component UUID to remove
- **recursive** (*bool, optional*) – Searches recursively, defaults to False
- **with_groups** (*bool, optional*) – Include components in groups, defaults to True

Returns Success of search and removal

Return type bool

search_by_name(*name: str*, *recursive: bool = False*, *case_sensitive: bool = False*) → list[*pyapi_rts.api.component.Component*] | None

Searches for components by their name

Parameters

- **name** (*str*) – Name to search for
- **recursive** (*bool, optional*) – Searches recursively in contained boxes, defaults to False
- **case_sensitive** (*bool, optional*) – Case sensitive search, defaults to False

Returns list of components with the given name

Return type list[*Component*]

set_parameter_at(*cid: str*, *param_key: str*, *value: Any*) → bool

Sets a parameter at the component with the given UUID

Parameters

- **cid** (*str*) – The component UUID
- **paramKey** (*str*) – The key of the parameter to set
- **value** (*Any*) – The value to set

Returns Success of operation

Return type bool

pyapi_rts.api.component_box.**add_xrack_connections**(*xrack_connections: dict*, *graph: networkx.classes.graph.Graph*, *mark_xrack: bool*) → None

pyapi_rts.api.draft module

class pyapi_rts.api.draft.**CompileMode**(*value*)

Bases: enum.Enum

An enumeration.

AUTO = 'AUTO'

PRIORITY = 'PRIORITY'

class pyapi_rts.api.draft.**Draft**(*version: str = '1.2'*, *title: str = 'Test Circuit'*, *author_created: str = 'pyapi_rts'*, *author_changed: str = 'pyapi_rts'*, *date_created: datetime.date = datetime.date(2023, 2, 28)*, *date_changed: datetime.date = datetime.date(2023, 2, 28)*, *time_step_us: float = 50.0*, *realtime:* pyapi_rts.api.draft.RealTime *= RealTime.Yes*, *non_rt_computation_us: int = 150*, *compile_mode:* pyapi_rts.api.draft.CompileMode *= CompileMode.AUTO*, *show_feedback_warnings: bool = False*, *circuit_comments: Optional[list[str]] = None*, *finish_time: float = 0.2*, *rack_number: int = 1*, *canvas_width: int = 1500*, *canvas_height: int = 850*, *subsys_index: int = 0*, *view_mode: int = 3*, *zoom: int = 100*, *top_left_point: tuple[int, int] = (0, 0)*)

Bases: object

RSCAD Draft, containing multiple subsystems

add_component(*component:* pyapi_rts.api.component.Component, *box_id: str*) → bool

Adds a component to the ComponentBox with the specified UUID/Index.

Parameters

- **component** (Component) – Component to add to the draft.
- **subsystem_id** (*str*) – The UUID or Subsystem index of the Component Box to add the component to.

Returns Boolean success

Return type bool

add_subsystem(*subsystem:* pyapi_rts.api.subsystem.Subsystem)

Adds a subsystem to the draft

Parameters **subsystem** (Subsystem) – Subsystem to add

generate_full_graph() → networkx.classes.graph.Graph

get_by_id(*cid: str*) → *pyapi_rts.api.component.Component* | None

Get a component from the draft by its id

Parameters **cid** (*str*) – Component UUID to search for

Returns Component if it is found, else None

Return type *Component* | None

get_components(*recursive: bool = True*, *clone=True*, *with_groups=False*) → list[*pyapi_rts.api.component.Component*]

Returns all components in the draft

Parameters **recursive** (*bool, optional*) – Include components from nested boxes, defaults to True

Returns list of components

Return type list[*Component*]

get_components_by_type(*type_name: str*, *recursive: bool = True*, *clone=True*, *with_groups=False*) → list[*pyapi_rts.api.component.Component*]

Returns all components of a given type in the draft

Parameters

- **type_name** (*str*) – Name of the component type
- **recursive** (*bool, optional*) – Recursive search, defaults to True

Returns list of components

Return type list[*Component*]

get_connection_graph() → networkx.classes.graph.Graph

Returns the combined connection graph from the subsystems.

Returns Combined connection graph

Return type Graph

get_rack_type() → int

Returns the rack type.

Returns Rack type

Return type int

get_rlc_tline(*name: str*) → *pyapi_rts.api.lark.rlc_tline.RLCTLine*

Returns the TLine Constants file as a RLC Tline.

Parameters **name** (*str*) – Name of the TLine file.

Returns RLC TLine

Return type *RLCTLine*

get_tline_constants(*name: str*) → *pyapi_rts.api.lark.tli_transformer.TliFile* | None

Search and returns the TLI file with the specified name.

Parameters **name** (*str*) – Name of the TLine Constants file.

Returns Tli file data as dicitonaries. None if not found.

Return type *TliFile* | None

modify_component(*component:* pyapi_rts.api.component.Component) → bool

Modifies a component in the draft if it exists.

Parameters **component** (Component) – The component to be modified.

Returns Boolean success

Return type bool

path: str

The path of the dfx file.

read_file(*path: str*)

Reads a .dfx file from the path and fills the object with the data

Parameters **path** (*str*) – Path to the .dfx file

remove_component(*cid: str*) → bool

Removes a component from the draft if it exists.

Parameters **cid** (*str*) – The UUID of the component to be removed.

Returns Boolean success

Return type bool

search_by_name(*name: str*, *recursive: bool = False*, *case_sensitive: bool = False*) → dict[str, list[*pyapi_rts.api.component.Component*]]

Search for components by name

Parameters

- **name** (*str*) – Name to search for
- **recursive** (*bool, optional*) – Recursive search, defaults to False
- **case_sensitive** (*bool, optional*) – Case sensitive search, defaults to False

Returns A mapping from the subsystem name to the list of found components

Return type dict[str, list[*Component*]]

property subsystems: list[*pyapi_rts.api.subsystem.Subsystem*]

Returns all subsystems in the draft

Returns list of subsystems

Return type list[*Subsystem*]

write_file(*path: str = ''*)

Writes the object to a .dfx file

Parameters **path** (*str*) – Path to the .dfx file

class pyapi_rts.api.draft.**RackType**(*value*)

Bases: enum.Enum

An enumeration.

GTWIF_GPC = 2

GTWIF_PB = 3

GTWIF_UNUSED = 0

NONE = -1

NOVACOR = 4

class pyapi_rts.api.draft.**RealTime**(*value*)

Bases: enum.Enum

An enumeration.

No = 'No'

Yes = 'Yes'

pyapi_rts.api.enumeration module

class pyapi_rts.api.enumeration.**Enumeration**

Bases: pyapi_rts.api.internals.dfxblock.DfxBlock

Enumeration settings for a component. There can be multiple enumerators in one file, but they work with internal UUIDs, not easy to reproduce.

apply(*name: str*) → str

Applies the rules of this enumeration to a string

Parameters **name** (*str*) – String to apply the rules to

Returns Modified copy of name with rules applied

Return type str

block() → list[str]

Returns the enumeration block of the .dfx file

Returns Enumeration block of the .dfx file

Return type list[str]

counter: dict = {}

enumeration_string: str

The enumeration string inserted into the name parameter.

is_active: bool

Is the enumeration feature activated?

read_block(*block: list[str]*, *name: str*)

Reads the enumeration block of the .dfx file

Parameters

- **block** (*list[str]*) – Enumeration block of the .dfx file
- **name** (*str*) – Type name of the component

style: *EnumerationStyle*

The style of the enumeration value.

value: int

The enumeration value as integer

property value_str: str

String representation with applied style of the enumeration value. :return: Enumeration value with applied style :rtype: str

class pyapi_rts.api.enumeration.**EnumerationStyle**(*value*)

Bases: str, enum.Enum

An enumeration.

Hex = 'Hex'

Integer = 'Integer'

lowercase = 'lowercase'

uppercase = 'uppercase'

pyapi_rts.api.group module

class pyapi_rts.api.group.**Group**

Bases: *pyapi_rts.api.component.Component*, *pyapi_rts.api.component_box.ComponentBox*

Group of components

as_dict() → dict[str, Any]

Returns the parameters of the component as a dictionary

Returns The parameters of the component as a dictionary

Return type dict[str, *Parameter*]

block() → list[str]

Writes the hierarchy to a .dfx block

Returns Hierarchy block of a .dfx file

Return type list[str]

get_box_type() → int

Returns the type of the component box. :return: The type of the component box. :rtype: int

get_by_key(*key: str*) → Optional[Any]

Returns the parameter with a certain key

Parameters

- **key** (*str*) – The key of the parameter
- **default_value** (*Any, optional*) – The default value if the parameter is not found, defaults to None

Returns The parameter or the default value if not found

Return type Any | None

has_key(*key: str*) → bool

Checks if a parameter with a certain key exists

Parameters **key** (*str*) – The key of the parameter

Returns True if the parameter exists, False otherwise

Return type bool

read_block(*block: pyapi_rts.api.internals.block.Block*, *check=True*)

Reads a component from a list of lines

Parameters

- **block** (*Block*) – A list of lines describing the component
- **check** (*bool, optional*) – Checks the block format before parsing, defaults to True

set_by_key(*key: str*, *value: Any*) → bool

Sets a parameter with a certain key

Parameters

- **key** (*str*) – The key of the parameter
- **value** (*Any*) – The value of the parameter

Returns True if the parameter was set successfully, False otherwise

Return type bool

pyapi_rts.api.hierarchy module

class pyapi_rts.api.hierarchy.**Hierarchy**

Bases: pyapi_rts.generated.HIERARCHY.HIERARCHY, *pyapi_rts.api.component_box.ComponentBox*

A component of type hierarchy, can contain other components

block() → list[str]

Writes the hierarchy to a .dfx block

Returns Hierarchy block of a .dfx file

Return type list[str]

get_box_type() → int

Returns the type of the box.

Returns Type of the box

Return type int

read_block(*block: pyapi_rts.api.internals.block.Block*, *check=True*)

Reads a hierarchy block of a .dfx file

Parameters **block** (*Block*) – Hierarchy block of a .dfx file

pyapi_rts.api.subsystem module

class pyapi_rts.api.subsystem.**Subsystem**(*draft*, *number: int*, *canvas_size_x: int = 3000*, *canvas_size_y: int = 2000*, *print_layout: pyapi_rts.api.subsystem.SubsystemPrintLayout = SubsystemPrintLayout.PORTRAIT*, *paper_type: pyapi_rts.api.subsystem.SubsystemPaperType = SubsystemPaperType.LETTER*)

Bases: pyapi_rts.api.internals.dfxblock.DfxBlock, *pyapi_rts.api.component_box.ComponentBox*

RSCAD subsystem, a canvas with components on it

block() → list[str]

Writes the subsystem to a .dfx file

Returns A list of strings representing the subsystem block

Return type list[str]

property index: str

The index of the subsystem in the draft.

Returns The index of the subsystem in the draft as a string.

Return type str

read_block(*block: list[str]*)

Read a subsystem block from a DFX file

Parameters **block** (*list[str]*) – A subsystem block

class pyapi_rts.api.subsystem.**SubsystemPaperType**(*value*)

Bases: enum.Enum

An enumeration.

A3 = 'A3'

A4 = 'A4'

A5 = 'A5'

ANSI_C = 'ANSI_C'

ANSI_D = 'ANSI_D'

ANSI_E = 'ANSI_E'

B4 = 'B4'

LEDGER = 'LEDGER'

LEGAL = 'LEGAL'

LETTER = 'LETTER'

W11_7_H17 = 'W11_7_H17'

class pyapi_rts.api.subsystem.**SubsystemPrintLayout**(*value*)

Bases: enum.Enum

An enumeration.

LANDSCAPE = 'LANDSCAPE'

PORTRAIT = 'PORTRAIT'

Module contents

api can read, edit and write network models. in the .dfx format used by RSCAD FX.

class pyapi_rts.api.**Component**(*type_name: Optional[str] = None*, *stretchable:* pyapi_rts.shared.stretchable.Stretchable *= Stretchable.NO*, *linked: Optional[bool] = None*)

Bases: pyapi_rts.api.internals.dfxblock.DfxBlock

A RSCAD component

GRID_SIZE = 32

LOAD_UNITS_DEFAULT = 10

LOAD_UNIT_NAMES = ['loadunit', 'LoadUnit', 'load']

abstract as_dict() → dict[str, *pyapi_rts.api.parameters.parameter.Parameter*]

Returns the parameters of the component as a dictionary

Returns The parameters of the component as a dictionary

Return type dict[str, *Parameter*]

block() → list[str]

Returns the component as a .dfx block

Returns The component as a .dfx block

Return type list[str]

property bounding_box: tuple[int, int, int, int]

property bounding_box_abs: tuple[int, int, int, int]

bounding_box_from_dict(*dictionary: dict*, *absolute: bool = False*) → tuple[int, int, int, int]

property connection_points: dict[str, *pyapi_rts.api.parameters.connection_point.ConnectionPoint*]

connection_points_from_dict(*dictionary*) → dict[str, *pyapi_rts.api.parameters.connection_point.ConnectionPoint*]

duplicate() → *pyapi_rts.api.component.Component*

Creates a copy of the component with the same UUID

Returns The copy of the component

Return type *Component*

enumeration: *Enumeration*

generate_pos_dict() → dict[str, list[tuple]]

Creates a dictionary that maps positions to connection points Key: "{x-coord},{y-coord}" of connection point Value: tuple of name of connection point and id of component

Returns The created dictionary

Return type dict[str, list[tuple]]

get_by_key(*key: str*, *default_value: Optional[Any] = None*, *as_int: bool = False*) → Optional[Any]

Returns the parameter with a certain key

Parameters

- **key** (*str*) – The key of the parameter
- **default_value** (*Any, optional*) – The default value if the parameter is not found, defaults to None

Returns The parameter or the default value if not found

Return type Any | None

get_connected_at_point(*point_name: str*, *return_connecting: bool = False*, *component_type: Optional[str] = None*) → list['Component']

Returns a list of all components connected at the connection point with the given name. Filters for components of a given type if component_type is specified.

Parameters

- **point_name** (*str*) – Name of the connection point
- **return_connecting** (*bool, optional*) – Returns the connecting components if True, otherwise only the end components are returned, defaults to False
- **component_type** (*str, optional*) – The type of components to return, defaults to None

Returns list of connected components

Return type list[*Component*]

get_special_value(*key: str*) → str

Returns the special value of the component. :param key: The key of the special value. :type key: str :return: The special value or empty string if not found. :rtype: str

graph_similar_to(*comp:* pyapi_rts.api.component.Component) → bool

Checks if the two components are identical for the purposes of the graph reperesentation. That is the case if: 1. They have the same id 2. They have the same coordinates, mirror and rotation 3. They have the same rectangle position and size

Parameters **comp** (Component) – The component to check for similarity.

Returns True if the two components are identical (for the graph).

Return type bool

has_key(*key: str*) → bool

Checks if a parameter with a certain key exists

Parameters **key** (*str*) – The key of the parameter

Returns True if the parameter exists, False otherwise

Return type bool

property height: int

property is_connecting: bool

Whether the component is a connecting component like wire or bus

Returns Whether the component is connecting

Return type bool

property is_hierarchy_connecting: bool

Whether the component is a connecting hierarchies without being a component box.

Returns Whether the component is hierarchy connecting

Return type bool

property is_label: bool

Whether the component is a label

Returns Whether the component is a label

Return type bool

linked: bool

property load_units: int

Returns the load units of the component based on the data available. :return: The load units of the component. :rtype: int

property mirror: int

The mirror state of the component

Returns The mirror of the component (0: no mirror, 1: mirror)

Return type int

property name: str

The parameter with key 'Name' with the enumerator applied

overlaps(*other:* pyapi_rts.api.component.Component) → bool

Checks if the rectangles overlap.

Parameters **other** (Component) – Another component

Returns True if the rectangles overlap, False otherwise

Return type bool

parent: *ComponentBox*

The component that contains this component.

read_block(*block: pyapi_rts.api.internals.block.Block*, *check=True*)

Reads a component from a list of lines

Parameters

- **block** (*Block*) – A list of lines describing the component
- **check** (*bool, optional*) – Checks the block format before parsing, defaults to True

property rotation: int

The rotation of the component

Returns The rotation of the component (times 90 degrees)

Return type int

set_by_key(*key: str*, *value: Any*) → bool

Sets a parameter with a certain key

Parameters

- **key** (*str*) – The key of the parameter
- **value** (*Any*) – The value of the parameter

Returns True if the parameter was set successfully, False otherwise

Return type bool

stretchable: *Stretchable*

Stretchable dimensions of the component

touches(*comp:* pyapi_rts.api.component.Component) → list[tuple[*pyapi_rts.api.parameters.connection_point.ConnectionPoint*, *pyapi_rts.api.parameters.connection_point.ConnectionPoint*]]

Returns a list of connection points the two components touch at the same time

Parameters **comp** (Component) – The component to check for touching connection points

Returns A list of connection points the two components touch at the same time

Return type list[tuple[*ConnectionPoint*, *ConnectionPoint*]]

property type: str

The component type

Returns The component type

Return type str

property uuid: str

Returns the component uuid

Returns The component UUID

Return type str

property width: int

property x: int

The x coordinate of the component

Returns The x coordinate of the component

Return type int

property x1: int

property x2: int

property y: int

The y coordinate of the component

Returns The y coordinate of the component

Return type int

property y1: int

property y2: int

class pyapi_rts.api.**ComponentBox**(*parent=None*)

Bases: object

Abstract class for an object containing a list of components

add_component(*component:* pyapi_rts.api.component.Component) → None

Add a component to the component box and update the connection graph and other data structures.

Parameters **component** (Component) – The component to add to this box

box_parent

The parent component box of this component box

generate_full_graph() → tuple[networkx.classes.graph.Graph, dict]

Generate the full graph consisting of the union of all componentBoxes included in this one.

Returns The graph and dictionary of cross-hierarchy connection points.

Return type tuple[Graph, dict]

get_at_point(*uuid: str*, *point_name: str*) → list[tuple[str, str]]

Returns a list of connection points at a given position on the grid.

Parameters

- **uuid** (*str*) – The UUID of the component to search from.

- **point_name** (*str*) – The name of the connection point to search from.

Returns list of (uuid, point_name) tuples.

Return type list[tuple[str, str]]

get_box_type() → int

Returns the type of the component box. :return: The type of the component box. :rtype: int

get_by_id(*cid: str*, *recursive: bool = True*, *with_groups=True*) → *pyapi_rts.api.component.Component* | None

Get a component by its id

Parameters

- **cid** (*str*) – Component UUID to search for
- **recursive** (*bool, optional*) – Searches recursively in boxes, defaults to True
- **with_groups** (*bool, optional*) – Include components in groups, defaults to True

Returns Component with the given UUID if found, None otherwise

Return type *Component* | None

get_component_boxes(*recursive: bool = False*) → list['ComponentBox']

Returns a list of all component boxes in the component box.

get_components(*recursive=False*, *clone=True*, *with_groups=False*) → list[*pyapi_rts.api.component.Component*]

Returns a list of all components in the component box.

Parameters

- **recursive** (*bool, optional*) – Also lists components in component boxes contained in this, defaults to False.
- **copy** (*bool, optional*) – Returns a copy of the list instead of the list itself, defaults to True
- **with_groups** (*bool, optional*) – Include components in groups, defaults to False

Returns list of components in the component box

Return type list[*Component*]

get_connected_at_component_point(*uuid: str*, *point_name: str*, *return_connecting: bool = False*, *component_type: Optional[str] = None*, *callers: list['ComponentBox'] = []*) → list[*pyapi_rts.api.component.Component*]

Returns a list of all components connected at the connection point with the given name. Filters for components of a given type if component_type is specified.

Parameters

- **point_name** (*str*) – Name of the connection point
- **component_type** (*str or None optional*) – Only return components of this type, defaults to None
- **callers** (*list[ComponentBox] optional*) – list of components that have already been called, defaults to []

Returns list of all components connected to the given label

Return type list[*Component*]

get_connected_to(*component:* pyapi_rts.api.component.Component, *clone: bool = True*, *include_all_connections: bool = False*) → list[*pyapi_rts.api.component.Component*]

Returns all components connected to a certain component, including those from hierarchies

Parameters

- **component** (Component) – Initial component to search from
- **clone** (*bool, optional*) – Whether to clone the components, defaults to True
- **include_all_connections** (*bool, optional*) – Whether to include non-signal connections, e.g. TLINE to calc block.

Returns list of all components connected to the given component

Return type list[*Component*]

get_connected_to_label(*label_name: str*, *return_connecting: bool = False*, *callers=[]*) → list[*pyapi_rts.api.component.Component*]

Returns all components connected to a wire or bus with a label with the given name. Returns the empty list if the label does not exist.

Parameters

- **label_name** (*str*) – The label of the bus or wire connection
- **return_connecting** (*bool*) – If true, returns the connecting components.
- **callers** (*list[ComponentBox]*) – list of ComponentBoxes that have already been called.

Returns list of all components connected to the given label

Return type list[*Component*]

get_connection_graph() → networkx.classes.graph.Graph

Returns the connection graph and generates it if it is not already generated.

The connection graph only contains connections in the same hierarchy level and does not include connections via wire label. This method also triggers the generation of the link dictionary.

Returns The connection graph

Return type Graph

get_draft()

Returns the draft of the component box.

Returns The draft this component box is part of

Return type *pyapi_rts.api.draft.Draft*

get_groups() → list['ComponentBox']

Returns a list of all groups in the component box.

Returns list of groups in the component box

Return type list[*Group*]

get_hierarchies(*recursive=False*) → list[*pyapi_rts.api.component.Component*]

Returns all hierarchy components in the component box

Parameters **recursive** (*bool, optional*) – Recusive search, defaults to False

Returns list of all hierarchies in the component box

Return type list[*Component*]

get_link_dict() → dict[str, list[tuple[str, str, *pyapi_rts.shared.node_type.NodeType*]]]

Returns the link dictionary and generates it if it is not already generated.

The link dictionary links the name of a connection point to a list of component UUIDs. It only includes NAME_CONNECTED connection points, e.g. of bus labels and wire labels.

Returns The link dictionary; (Component.uuid, ConnectionPoint.name, ConnectionPoint.link_type)

Return type dict[str, list[tuple[str, str, *NodeType*]]]

get_rack_type() → int

Returns the rack type of the component box. :return: The rack type of the component box. :rtype: int

modify_component(*component:* pyapi_rts.api.component.Component, *recursive=True*) → bool

Modify a component in the component box and update the connection graph and other data structures.

Parameters

- **component** (Component) – The component to modify
- **recursive** (*bool, optional*) – Searches recursively, defaults to True

Returns Success of search and modification

Return type bool

remove_component(*cid: str*, *recursive: bool = False*, *with_groups=True*) → bool

Remove a component from the component box and update the connection graph and other data structures.

Parameters

- **cid** (*str*) – Component UUID to remove
- **recursive** (*bool, optional*) – Searches recursively, defaults to False
- **with_groups** (*bool, optional*) – Include components in groups, defaults to True

Returns Success of search and removal

Return type bool

search_by_name(*name: str*, *recursive: bool = False*, *case_sensitive: bool = False*) → list[*pyapi_rts.api.component.Component*] | None

Searches for components by their name

Parameters

- **name** (*str*) – Name to search for
- **recursive** (*bool, optional*) – Searches recursively in contained boxes, defaults to False
- **case_sensitive** (*bool, optional*) – Case sensitive search, defaults to False

Returns list of components with the given name

Return type list[*Component*]

set_parameter_at(*cid: str*, *param_key: str*, *value: Any*) → bool

Sets a parameter at the component with the given UUID

Parameters

- **cid** (*str*) – The component UUID
- **paramKey** (*str*) – The key of the parameter to set
- **value** (*Any*) – The value to set

Returns Success of operation

Return type bool

class pyapi_rts.api.**Draft**(*version: str = '1.2'*, *title: str = 'Test Circuit'*, *author_created: str = 'pyapi_rts'*, *author_changed: str = 'pyapi_rts'*, *date_created: datetime.date = datetime.date(2023, 2, 28)*, *date_changed: datetime.date = datetime.date(2023, 2, 28)*, *time_step_us: float = 50.0*, *realtime:* pyapi_rts.api.draft.RealTime *= RealTime.Yes*, *non_rt_computation_us: int = 150*, *compile_mode:* pyapi_rts.api.draft.CompileMode *= CompileMode.AUTO*, *show_feedback_warnings: bool = False*, *circuit_comments: Optional[list[str]] = None*, *finish_time: float = 0.2*, *rack_number: int = 1*, *canvas_width: int = 1500*, *canvas_height: int = 850*, *subsys_index: int = 0*, *view_mode: int = 3*, *zoom: int = 100*, *top_left_point: tuple[int, int] = (0, 0)*)

Bases: `object`

RSCAD Draft, containing multiple subsystems

add_component(*component:* pyapi_rts.api.component.Component, *box_id: str*) → bool

Adds a component to the ComponentBox with the specified UUID/Index.

Parameters

- **component** (Component) – Component to add to the draft.
- **subsystem_id** (*str*) – The UUID or Subsystem index of the Component Box to add the component to.

Returns Boolean success

Return type bool

add_subsystem(*subsystem:* pyapi_rts.api.subsystem.Subsystem)

Adds a subsystem to the draft

Parameters **subsystem** (Subsystem) – Subsystem to add

generate_full_graph() → networkx.classes.graph.Graph

get_by_id(*cid: str*) → *pyapi_rts.api.component.Component* | None

Get a component from the draft by its id

Parameters **cid** (*str*) – Component UUID to search for

Returns Component if it is found, else None

Return type *Component* | None

get_components(*recursive: bool = True*, *clone=True*, *with_groups=False*) → list[*pyapi_rts.api.component.Component*]

Returns all components in the draft

Parameters **recursive** (*bool, optional*) – Include components from nested boxes, defaults to True

Returns list of components

Return type list[*Component*]

get_components_by_type(*type_name: str*, *recursive: bool = True*, *clone=True*, *with_groups=False*) → list[*pyapi_rts.api.component.Component*]

Returns all components of a given type in the draft

Parameters

- **type_name** (*str*) – Name of the component type
- **recursive** (*bool, optional*) – Recursive search, defaults to True

Returns list of components

Return type list[*Component*]

get_connection_graph() → networkx.classes.graph.Graph

Returns the combined connection graph from the subsystems.

Returns Combined connection graph

Return type Graph

get_rack_type() → int

Returns the rack type.

Returns Rack type

Return type int

get_rlc_tline(*name: str*) → *pyapi_rts.api.lark.rlc_tline.RLCTLine*

Returns the TLine Constants file as a RLC Tline.

Parameters **name** (*str*) – Name of the TLine file.

Returns RLC TLine

Return type *RLCTLine*

get_tline_constants(*name: str*) → *pyapi_rts.api.lark.tli_transformer.TliFile* | None

Search and returns the TLI file with the specified name.

Parameters **name** (*str*) – Name of the TLine Constants file.

Returns Tli file data as dicitonaries. None if not found.

Return type *TliFile* | None

modify_component(*component:* pyapi_rts.api.component.Component) → bool

Modifies a component in the draft if it exists.

Parameters **component** (Component) – The component to be modified.

Returns Boolean success

Return type bool

path: str

The path of the dfx file.

rack_types: list[*pyapi_rts.api.draft.RackType*]

read_file(*path: str*)

Reads a .dfx file from the path and fills the object with the data

Parameters **path** (*str*) – Path to the .dfx file

remove_component(*cid: str*) → bool

Removes a component from the draft if it exists.

Parameters **cid** (*str*) – The UUID of the component to be removed.

Returns Boolean success

Return type bool

search_by_name(*name: str*, *recursive: bool = False*, *case_sensitive: bool = False*) → dict[str, list[*pyapi_rts.api.component.Component*]]

Search for components by name

Parameters

- **name** (*str*) – Name to search for
- **recursive** (*bool, optional*) – Recursive search, defaults to False
- **case_sensitive** (*bool, optional*) – Case sensitive search, defaults to False

Returns A mapping from the subsystem name to the list of found components

Return type dict[str, list[*Component*]]

property subsystems: list[*pyapi_rts.api.subsystem.Subsystem*]

Returns all subsystems in the draft

Returns list of subsystems

Return type list[*Subsystem*]

write_file(*path: str = ''*)

Writes the object to a .dfx file

Parameters **path** (*str*) – Path to the .dfx file

class pyapi_rts.api.**Enumeration**

Bases: pyapi_rts.api.internals.dfxblock.DfxBlock

Enumeration settings for a component. There can be multiple enumerators in one file, but they work with internal UUIDs, not easy to reproduce.

apply(*name: str*) → str

Applies the rules of this enumeration to a string

Parameters **name** (*str*) – String to apply the rules to

Returns Modified copy of name with rules applied

Return type str

block() → list[str]

Returns the enumeration block of the .dfx file

Returns Enumeration block of the .dfx file

Return type list[str]

counter: dict = {}

enumeration_string: str

The enumeration string inserted into the name parameter.

is_active: bool

Is the enumeration feature activated?

read_block(*block: list[str]*, *name: str*)

Reads the enumeration block of the .dfx file

Parameters

- **block** (*list[str]*) – Enumeration block of the .dfx file
- **name** (*str*) – Type name of the component

style: *EnumerationStyle*

The style of the enumeration value.

value: int

The enumeration value as integer

property value_str: str

String representation with applied style of the enumeration value. :return: Enumeration value with applied style :rtype: str

class pyapi_rts.api.**Hierarchy**

Bases: pyapi_rts.generated.HIERARCHY.HIERARCHY, *pyapi_rts.api.component_box.ComponentBox*

A component of type hierarchy, can contain other components

block() → list[str]

Writes the hierarchy to a .dfx block

Returns Hierarchy block of a .dfx file

Return type list[str]

get_box_type() → int

Returns the type of the box.

Returns Type of the box

Return type int

read_block(*block: pyapi_rts.api.internals.block.Block*, *check=True*)

Reads a hierarchy block of a .dfx file

Parameters **block** (*Block*) – Hierarchy block of a .dfx file

class pyapi_rts.api.**Subsystem**(*draft*, *number: int*, *canvas_size_x: int = 3000*, *canvas_size_y: int = 2000*, *print_layout:* pyapi_rts.api.subsystem.SubsystemPrintLayout = *SubsystemPrintLayout.PORTRAIT*, *paper_type:* pyapi_rts.api.subsystem.SubsystemPaperType = *SubsystemPaperType.LETTER*)

Bases: pyapi_rts.api.internals.dfxblock.DfxBlock, *pyapi_rts.api.component_box.ComponentBox*

RSCAD subsystem, a canvas with components on it

block() → list[str]

Writes the subsystem to a .dfx file

Returns A list of strings representing the subsystem block

Return type list[str]

property index: str

The index of the subsystem in the draft.

Returns The index of the subsystem in the draft as a string.

Return type str

read_block(*block: list[str]*)

Read a subsystem block from a DFX file

Parameters **block** (*list[str]*) – A subsystem block

pyapi_rts.class_extractor package

Subpackages

pyapi_rts.class_extractor.extracted package

Submodules

pyapi_rts.class_extractor.extracted.comp_def_parameter module

class pyapi_rts.class_extractor.extracted.comp_def_parameter.**CompDefParameter**(*key*, *description*, *descValid*, *mystery*, *_type*, *default*, *_from*, *_to*, *_if*)

Bases: object

A parameter of a component read from the definition file

as_ext_parameter() → tuple[*pyapi_rts.class_extractor.extracted.ext_parameter.ExtParameter*, *pyapi_rts.class_extractor.extracted.ext_enum_parameter.ExtEnumParameter* | None]

Converts the parameter to an ExtParameter maybe the dependent ExtEnumParameter

Raises **Exception** – Type of parameter not supported

Returns ExtParameter and ExtEnumParameter if dependent

Return type tuple[*ExtParameter*, *ExtEnumParameter* | None]

property comp_type

The type of the component

default

The default value of the parameter

description

The description of the parameter

key

The key of the parameter

mystery

A number with undetermined purpose

pyapi_rts.class_extractor.extracted.ext_component module

class pyapi_rts.class_extractor.extracted.ext_component.**ExtComponent**

Bases: object

A representation of the component for conversion between other formats

apply_tag_dict(*tag_dict: list[str]*) → None

Applies a list of tags to the relevant attributes

Parameters **tag_dict** (*list[str]*) – A dictionary of lists of tags, generated by the ClassExtractor from the component_tags file

collections: list[*pyapi_rts.class_extractor.extracted.ext_parameter_coll.ExtParameterColl*]

The parameter collections of the component

computations: list[tuple[str, str, str]]

Computations: (name, type, expression)

is_connecting: bool

True if this component is a wire, bus or similar

is_hierarchy_connecting: bool

True if the component can connect component boxes without being one of its own

is_label: bool

True if the component is a label

name_parameter_key: str

The parameter determining the name of the component

parameters: list[*pyapi_rts.class_extractor.extracted.ext_parameter.ExtParameter*]

The top-level parameters of the component

read(*_list: list[str]*) → None

Loads the component from a list of lines

Parameters **_list** (*list[str]*) – list of lines

rectangle: *pyapi_rts.class_extractor.extracted.ext_rectangle.ExtRectangle* | None

The surrounding rectangle of the component, including conneciton points

set_type(*_type: str*) → None

Sets the component type

Parameters **_type** (*str*) – The component type

property type

The type of the component

Returns The type of the component

Return type str

property type_name

The name of the component type

Returns The name of the component type

Return type str

write() → list[str]

Converts the component to a list of lines :return: list of lines :rtype: list[str]

pyapi_rts.class_extractor.extracted.ext_connection_point module

class pyapi_rts.class_extractor.extracted.ext_connection_point.**ExtConnectionPoint**

Bases: object

A connection point of a rectangle.

component_init() → str

Returns the component initialization code in Python.

Returns The component initialization code.

Return type str

link_name: str

Link Name of the connection point.

merge(*other:* pyapi_rts.class_extractor.extracted.ext_connection_point.ExtConnectionPoint) → None

Merges the node with another node.

Parameters **other** (ExtConnectionPoint) – Other node

x: int | str

X position of the connection point relative to the center.

y: int | str

Y position of the connection point relative to the center.

pyapi_rts.class_extractor.extracted.ext_enum_parameter module

class pyapi_rts.class_extractor.extracted.ext_enum_parameter.**ExtEnumParameter**(*name: str*)

Bases: object

A parameter that contains an enumeration

property enum_type: str

The type of the parameter

Returns Type of the parameter

Return type str

property name: str

Name of the parameter

Returns Name of the parameter

Return type str

options: list[str]

Available options for the value of the parameter

property options_hash: int

Hash over the options in their specific order

Returns Hash

Return type int

classmethod read(*lst: list[str]*) → *pyapi_rts.class_extractor.extracted.ext_enum_parameter.ExtEnumParameter*

Reads the parameter from a list of lines

Parameters **lst** (*list[str]*) – list of lines

Returns Read EnumParameter

Return type *ExtEnumParameter*

set_name(*name: str*) → None

Sets the name of the parameter

Parameters **name** (*str*) – Name of the parameter

write() → list[str]

Writes the parameter to a list of lines

Returns list of lines

Return type list[str]

pyapi_rts.class_extractor.extracted.ext_parameter module

class pyapi_rts.class_extractor.extracted.ext_parameter.**ExtParameter**(*key: str*, *name: str*, *_type: str*, *default: Any*, *description: str = ''*)

Bases: object

An intermediate parameter

property comp_type

The type of the component

default: Any

Default value of the parameter

description: str

Description of the parameter

key: str

Key of the parameter

name: str

Name of the parameter

classmethod read(*line: list[str]*) → *pyapi_rts.class_extractor.extracted.ext_parameter.ExtParameter*

Read the parameter from a list of lines

Parameters **line** (*list[str]*) – list of lines

Returns Read parameter

Return type *ExtParameter*

set_type(*_type: str*) → None

Set the type of the parameter

Parameters **_type** (*str*) – Type of the parameter

write() → list[str]

Write the parameter to a list of lines

Returns list of lines

Return type list[str]

pyapi_rts.class_extractor.extracted.ext_parameter_coll module

class pyapi_rts.class_extractor.extracted.ext_parameter_coll.**ExtParameterColl**(*name: str*)

Bases: object

A named collection of parameters.

name

The name of the collection.

parameters: list[*pyapi_rts.class_extractor.extracted.ext_parameter.ExtParameter*]

The parameters in the collection.

classmethod read(*lst: list[str]*) → Any

Reads an ExtParameterColl object from a list of strings.

Parameters **lst** (*list[str]*) – The list of strings.

Returns The ExtParameterColl object.

Return type *ExtParameterColl*

property type_name

The type name of the ExtParameterColl object.

Returns The type name.

Return type str

write() → list[str]

Writes the ExtParameterColl object to a list of strings.

Returns Lines of strings.

Return type list[str]

pyapi_rts.class_extractor.extracted.ext_rectangle module

class pyapi_rts.class_extractor.extracted.ext_rectangle.**ExtRectangle**

Bases: `object`

A rectangle around a RSCAD component with a given position, width and height.

component_init() → list[str]

Returns the component initialization code in Python.

Returns The component initialization code.

Return type list[str]

connection_points: list[*pyapi_rts.shared.condition_tree.ConditionTreeNode*]

Condition tree for the connection points.

graphics: list[*pyapi_rts.shared.condition_tree.ConditionTreeNode*]

Condition tree for the graphics instructions.

linked: bool

Component is linked to one or more other components.

rectangle_functions() → list[str]

Returns the rectangle functions in Python.

Returns The rectangle functions as Python code.

Return type list[str]

stretchable: *pyapi_rts.shared.stretchable.Stretchable*

Stretchable type.

write_lines() → list[str]

Writes the ExtRectangle object to a list of strings.

Returns Lines of strings.

Return type list[str]

Module contents

class pyapi_rts.class_extractor.extracted.**CompDefParameter**(*key*, *description*, *descValid*, *mystery*, *_type*, *default*, *_from*, *_to*, *_if*)

Bases: `object`

A parameter of a component read from the definition file

as_ext_parameter() → tuple[*pyapi_rts.class_extractor.extracted.ext_parameter.ExtParameter*, *pyapi_rts.class_extractor.extracted.ext_enum_parameter.ExtEnumParameter* | None]

Converts the parameter to an ExtParameter maybe the dependent ExtEnumParameter

Raises **Exception** – Type of parameter not supported

Returns ExtParameter and ExtEnumParameter if dependent

Return type tuple[*ExtParameter*, *ExtEnumParameter* | None]

property comp_type

The type of the component

default

The default value of the parameter

description

The description of the parameter

key

The key of the parameter

mystery

A number with undetermined purpose

class pyapi_rts.class_extractor.extracted.**ExtComponent**

Bases: object

A representation of the component for conversion between other formats

apply_tag_dict(*tag_dict: list[str]*) → None

Applies a list of tags to the relevant attributes

Parameters **tag_dict** (*list[str]*) – A dictionary of lists of tags, generated by the ClassExtractor from the component_tags file

collections:
list[*pyapi_rts.class_extractor.extracted.ext_parameter_coll.ExtParameterColl*]

The parameter collections of the component

computations: list[tuple[str, str, str]]

Computations: (name, type, expression)

is_connecting: bool

True if this component is a wire, bus or similar

is_hierarchy_connecting: bool

True if the component can connect component boxes without being one of its own

is_label: bool

True if the component is a label

name_parameter_key: str

The parameter determining the name of the component

parameters: list[*pyapi_rts.class_extractor.extracted.ext_parameter.ExtParameter*]

The top-level parameters of the component

read(*_list: list[str]*) → None

Loads the component from a list of lines

Parameters **_list** (*list[str]*) – list of lines

rectangle: *pyapi_rts.class_extractor.extracted.ext_rectangle.ExtRectangle* | None

The surrounding rectangle of the component, including conneciton points

set_type(*_type: str*) → None

Sets the component type

Parameters **_type** (*str*) – The component type

property type

The type of the component

Returns The type of the component

Return type str

property type_name

The name of the component type

Returns The name of the component type

Return type str

write() → list[str]

Converts the component to a list of lines :return: list of lines :rtype: list[str]

class pyapi_rts.class_extractor.extracted.**ExtConnectionPoint**

Bases: object

A connection point of a rectangle.

component_init() → str

Returns the component initialization code in Python.

Returns The component initialization code.

Return type str

link_name: str

Link Name of the connection point.

merge(*other:* pyapi_rts.class_extractor.extracted.ext_connection_point.ExtConnectionPoint) → None

Merges the node with another node.

Parameters **other** (ExtConnectionPoint) – Other node

name: str

phase: float

x: int | str

X position of the connection point relative to the center.

y: int | str

Y position of the connection point relative to the center.

class pyapi_rts.class_extractor.extracted.**ExtEnumParameter**(*name: str*)

Bases: object

A parameter that contains an enumeration

property enum_type: str

The type of the parameter

Returns Type of the parameter

Return type str

property name: str

Name of the parameter

Returns Name of the parameter

Return type str

options: list[str]

Available options for the value of the parameter

property options_hash: int

Hash over the options in their specific order

Returns Hash

Return type int

classmethod read(*lst: list[str]*) → *pyapi_rts.class_extractor.extracted.ext_enum_parameter.ExtEnumParameter*

Reads the parameter from a list of lines

Parameters **lst** (*list[str]*) – list of lines

Returns Read EnumParameter

Return type *ExtEnumParameter*

set_name(*name: str*) → None

Sets the name of the parameter

Parameters **name** (*str*) – Name of the parameter

write() → list[str]

Writes the parameter to a list of lines

Returns list of lines

Return type list[str]

class pyapi_rts.class_extractor.extracted.**ExtParameter**(*key: str*, *name: str*, *_type: str*, *default: Any*, *description: str = ''*)

Bases: object

An intermediate parameter

property comp_type

The type of the component

default: Any

Default value of the parameter

description: str

Description of the parameter

key: str

Key of the parameter

name: str

Name of the parameter

classmethod read(*line: list[str]*) → *pyapi_rts.class_extractor.extracted.ext_parameter.ExtParameter*

Read the parameter from a list of lines

Parameters **line** (*list[str]*) – list of lines

Returns Read parameter

Return type *ExtParameter*

set_type(*_type: str*) → None

Set the type of the parameter

Parameters **_type** (*str*) – Type of the parameter

write() → list[str]

Write the parameter to a list of lines

Returns list of lines

Return type list[str]

class pyapi_rts.class_extractor.extracted.**ExtParameterColl**(*name: str*)

Bases: object

A named collection of parameters.

name

The name of the collection.

parameters: list[*pyapi_rts.class_extractor.extracted.ext_parameter.ExtParameter*]

The parameters in the collection.

classmethod read(*lst: list[str]*) → Any

Reads an ExtParameterColl object from a list of strings.

Parameters **lst** (*list[str]*) – The list of strings.

Returns The ExtParameterColl object.

Return type *ExtParameterColl*

property type_name

The type name of the ExtParameterColl object.

Returns The type name.

Return type str

write() → list[str]

Writes the ExtParameterColl object to a list of strings.

Returns Lines of strings.

Return type list[str]

class pyapi_rts.class_extractor.extracted.**ExtRectangle**

Bases: object

A rectangle around a RSCAD component with a given position, width and height.

component_init() → list[str]

Returns the component initialization code in Python.

Returns The component initialization code.

Return type list[str]

connection_points: list[*pyapi_rts.shared.condition_tree.ConditionTreeNode*]

Condition tree for the connection points.

graphics: list[*pyapi_rts.shared.condition_tree.ConditionTreeNode*]

Condition tree for the graphics instructions.

linked: bool

Component is linked to one or more other components.

rectangle_functions() → list[str]

Returns the rectangle functions in Python.

Returns The rectangle functions as Python code.

Return type list[str]

stretchable: *pyapi_rts.shared.stretchable.Stretchable*

Stretchable type.

write_lines() → list[str]

Writes the ExtRectangle object to a list of strings.

Returns Lines of strings.

Return type list[str]

pyapi_rts.class_extractor.generators package

Submodules

pyapi_rts.class_extractor.generators.class_generator module

class pyapi_rts.class_extractor.generators.class_generator.**ClassGenerator**

Bases: `object`

A generator for a python class file with some helper functions and constants

BASIC_COMPONENTS = ['BooleanParameter', 'StringParameter', 'NameParameter', 'IntegerParameter', 'FloatParameter', 'ColorParameter']

BASIC_COMPONENT_PATH = 'pyapi_rts.api.parameters'

ENUM_PATH = 'pyapi_rts.generated.enums.'

GENERATED_PATH = 'pyapi_rts.generated.'

read_file(*path: pathlib.Path*) → list[str]

Reads the file

Parameters **path** (*Path*) – The path to the file

Returns The lines of the file

Return type list[str]

replace(*lines: list[str]*) → list[str]

Replaces the lines in the file with the generated lines

Parameters **lines** (*list[str]*) – The lines to replace

Returns The replaced lines

Return type list[str]

write_file(*path: pathlib.Path*, *lines: list[str]*) → bytes

Writes the lines to the file

Parameters

- **path** (*Path*) – The path to the file
- **lines** (*list[str]*) – The lines to write

Returns Hash of content of file

Return type bytes

pyapi_rts.class_extractor.generators.class_loader_generator module

class pyapi_rts.class_extractor.generators.class_loader_generator.**ClassLoaderGenerator**(*comps: list[*pyapi_rts.class_e
hook_names=list[str

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates the class loader responsible for loading all other classes at runtime

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – list of lines

Returns list of lines (changed)

Return type list[str]

replace_foreach(*line: str*) → list[str]

Replaces the FOREACH statement in one line

Parameters **line** (*str*) – Line to replace

Returns list of lines (changed)

Return type list[str]

replace_foreach_hook(*line: str*) → list[str]

Replaces the FOREACH_HOOK statement in one line

Parameters **line** (*str*) – Line to replace

Returns list of lines (changed)

Return type list[str]

pyapi_rts.class_extractor.generators.component_generator module

class pyapi_rts.class_extractor.generators.component_generator.**ComponentGenerator**(*comp:* pyapi_rts.class_extractor.ext

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates a python class representing a RSCAD FX component

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – Template file lines

Returns Changed lines

Return type list[str]

replace_foreach_coll(*line: str*) → list[str]

Replaces the FOREACH_COLL statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

replace_foreach_comp(*line: str*) → list[str]

Replaces the FOREACH_COMP statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

replace_foreach_param(*line: str*) → list[str]

Replaces the FOREACH_PARAM statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

replace_foreach_type(*line: str*) → list[str]

Replaces the FOREACH_TYPE statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

sanitize_parameter_name(*name: str*) → str

Sanitizes the parameter name to be valid Python variable name

Parameters **name** (*str*) – The parameter name

Returns The sanitized name

Return type str

pyapi_rts.class_extractor.generators.enum_generator module

class pyapi_rts.class_extractor.generators.enum_generator.**EnumGenerator**(*enum:* pyapi_rts.class_extractor.extracted.ext_er

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates a python class file from an ExtEnumParameter

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – Template file lines

Returns Template file lines (changed)

Return type list[str]

replace_foreach(*line: str*) → list[str]

Replaces the FOREACH statement in one line

Parameters **line** (*str*) – Line to replace

Returns Changed lines

Return type list[str]

pyapi_rts.class_extractor.generators.graphics_macro_generator module

class pyapi_rts.class_extractor.generators.graphics_macro_generator.**GraphicsMacroGenerator**(*bboxes*)

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates a dictionary with regular expressions for graphics macros.

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – Template file lines

Returns Changed lines

Return type list[str]

replace_foreach_func(*line: str*) → list[str]

Replaces the FOREACH_FUNC statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

replace_foreach_regex(*line: str*) → list[str]

Replaces the FOREACH_REGEX statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

pyapi_rts.class_extractor.generators.parameter_collection_generator module

class pyapi_rts.class_extractor.generators.parameter_collection_generator.**ParameterCollectionGenerator**(

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates a ParameterCollection (group of parameters) form an ExtParameterColl

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – Lines in template file

Returns Lines with replaced template statements

Return type list[str]

replace_foreach(*line: str*) → list[str]

Replaces the FOREACH statement in one line

Parameters **line** (*str*) – Line to replace

Returns list of lines (changed)

Return type list[str]

replace_foreachType(*line: str*) → list[str]

Replaces the FOREACH_TYPE statement in one line

Parameters **line** (*str*) – Line to replace

Returns list of lines (changed)

Return type list[str]

Module contents

Code generators for Python classes

class pyapi_rts.class_extractor.generators.**ClassGenerator**

Bases: object

A generator for a python class file with some helper functions and constants

BASIC_COMPONENTS = ['BooleanParameter', 'StringParameter', 'NameParameter', 'IntegerParameter', 'FloatParameter', 'ColorParameter']

BASIC_COMPONENT_PATH = 'pyapi_rts.api.parameters'

ENUM_PATH = 'pyapi_rts.generated.enums.'

GENERATED_PATH = 'pyapi_rts.generated.'

read_file(*path: pathlib.Path*) → list[str]

Reads the file

Parameters **path** (*Path*) – The path to the file

Returns The lines of the file

Return type list[str]

replace(*lines: list[str]*) → list[str]

Replaces the lines in the file with the generated lines

Parameters **lines** (*list[str]*) – The lines to replace

Returns The replaced lines

Return type list[str]

write_file(*path: pathlib.Path*, *lines: list[str]*) → bytes

Writes the lines to the file

Parameters

- **path** (*Path*) – The path to the file
- **lines** (*list[str]*) – The lines to write

Returns Hash of content of file

Return type bytes

class pyapi_rts.class_extractor.generators.**ClassLoaderGenerator**(*comps: list[*pyapi_rts.class_extractor.extracted.ext_compone *hook_names=list[str]*)

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates the class loader responsible for loading all other classes at runtime

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – list of lines

Returns list of lines (changed)

Return type list[str]

replace_foreach(*line: str*) → list[str]

Replaces the FOREACH statement in one line

Parameters **line** (*str*) – Line to replace

Returns list of lines (changed)

Return type list[str]

replace_foreach_hook(*line: str*) → list[str]

Replaces the FOREACH_HOOK statement in one line

Parameters **line** (*str*) – Line to replace

Returns list of lines (changed)

Return type list[str]

class pyapi_rts.class_extractor.generators.**ComponentGenerator**(*comp:* pyapi_rts.class_extractor.extracted.ext_component.Ext

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates a python class representing a RSCAD FX component

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – Template file lines

Returns Changed lines

Return type list[str]

replace_foreach_coll(*line: str*) → list[str]

Replaces the FOREACH_COLL statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

replace_foreach_comp(*line: str*) → list[str]

Replaces the FOREACH_COMP statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

replace_foreach_param(*line: str*) → list[str]

Replaces the FOREACH_PARAM statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

replace_foreach_type(*line: str*) → list[str]

Replaces the FOREACH_TYPE statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

sanitize_parameter_name(*name: str*) → str

Sanitizes the parameter name to be valid Python variable name

Parameters **name** (*str*) – The parameter name

Returns The sanitized name

Return type str

class pyapi_rts.class_extractor.generators.**EnumGenerator**(*enum:* pyapi_rts.class_extractor.extracted.ext_enum_parameter.ExtE

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates a python class file from an ExtEnumParameter

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – Template file lines

Returns Template file lines (changed)

Return type list[str]

replace_foreach(*line: str*) → list[str]

Replaces the FOREACH statement in one line

Parameters **line** (*str*) – Line to replace

Returns Changed lines

Return type list[str]

class pyapi_rts.class_extractor.generators.**GraphicsMacroGenerator**(*bboxes*)

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates a dictionary with regular expressions for graphics macros.

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – Template file lines

Returns Changed lines

Return type list[str]

replace_foreach_func(*line: str*) → list[str]

Replaces the FOREACH_FUNC statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

replace_foreach_regex(*line: str*) → list[str]

Replaces the FOREACH_REGEX statement in the template line

Parameters **line** (*str*) – The line to replace

Returns The replaced lines

Return type list[str]

class pyapi_rts.class_extractor.generators.**ParameterCollectionGenerator**(*pc:* pyapi_rts.class_extractor.extracted.ext_pa

Bases: *pyapi_rts.class_extractor.generators.class_generator.ClassGenerator*

Generates a ParameterCollection (group of parameters) form an ExtParameterColl

replace(*lines: list[str]*) → list[str]

Replaces the template statements in the lines

Parameters **lines** (*list[str]*) – Lines in template file

Returns Lines with replaced template statements

Return type list[str]

replace_foreach(*line: str*) → list[str]

Replaces the FOREACH statement in one line

Parameters **line** (*str*) – Line to replace

Returns list of lines (changed)

Return type list[str]

replace_foreachType(*line: str*) → list[str]

Replaces the FOREACH_TYPE statement in one line

Parameters **line** (*str*) – Line to replace

Returns list of lines (changed)

Return type list[str]

pyapi_rts.class_extractor.hooks package

Submodules

pyapi_rts.class_extractor.hooks.LinkedNodeHook module

class pyapi_rts.class_extractor.hooks.LinkedNodeHook.**LinkedNodeHook**

Bases: *pyapi_rts.shared.component_hook.ComponentHook*

Connects nodes that have the "linkNode" property set to "yes".

classmethod graph_connections(*components: list*, *pos_dict: dict*, *link_dict: dict*) → list[tuple[str, str]]

Hook method.

pyapi_rts.class_extractor.hooks.SpecialValueHook module

class pyapi_rts.class_extractor.hooks.SpecialValueHook.**SpecialValueHook**

Bases: *pyapi_rts.shared.component_hook.ComponentHook*

A hook providing default values for some of the undocumented special values.

classmethod special_value(*component:* pyapi_rts.api.component.Component, *key: str*) → Optional[Any]

Adds new special values to components. :param component: Component to evaluate. :type component: Component :return: Value of the special key or None if it does not exist for this component. :rtype: Any | None

pyapi_rts.class_extractor.hooks.TLineHook module

class pyapi_rts.class_extractor.hooks.TLineHook.**TLineHook**

Bases: *pyapi_rts.shared.component_hook.ComponentHook*

Adds TLINE connections.

Parameters **ComponentHook** (*_type_*) – _description_

classmethod graph_connections(*components: list[*pyapi_rts.api.component.Component*]*, *pos_dict: dict*, *link_dict: dict*) → list[tuple[str, str]]

Hook method.

pyapi_rts.class_extractor.hooks.XrTrfHook module

class pyapi_rts.class_extractor.hooks.XrTrfHook.**XrTrfHook**

Bases: *pyapi_rts.shared.component_hook.ComponentHook*

A hook for Cross Rack Transformers

classmethod link_connections(*components: list*) → list[tuple[str, str, *pyapi_rts.shared.node_type.NodeType*]]

Adds entries to link_dict for Crossrack Transformers.

Module contents

class pyapi_rts.class_extractor.hooks.**LinkedNodeHook**

Bases: *pyapi_rts.shared.component_hook.ComponentHook*

Connects nodes that have the "linkNode" property set to "yes".

classmethod graph_connections(*components: list*, *pos_dict: dict*, *link_dict: dict*) → list[tuple[str, str]]

Hook method.

class pyapi_rts.class_extractor.hooks.**SpecialValueHook**

Bases: *pyapi_rts.shared.component_hook.ComponentHook*

A hook providing default values for some of the undocumented special values.

classmethod special_value(*component:* pyapi_rts.api.component.Component, *key: str*) → Optional[Any]

Adds new special values to components. :param component: Component to evaluate. :type component: Component :return: Value of the special key or None if it does not exist for this component. :rtype: Any | None

class pyapi_rts.class_extractor.hooks.**TLineHook**

Bases: *pyapi_rts.shared.component_hook.ComponentHook*

Adds TLINE connections.

Parameters **ComponentHook** (*_type_*) – _description_

classmethod graph_connections(*components: list[*pyapi_rts.api.component.Component*]*, *pos_dict: dict*, *link_dict: dict*) → list[tuple[str, str]]

Hook method.

class pyapi_rts.class_extractor.hooks.**XrTrfHook**

Bases: *pyapi_rts.shared.component_hook.ComponentHook*

A hook for Cross Rack Transformers

classmethod link_connections(*components: list*) → list[tuple[str, str, *pyapi_rts.shared.node_type.NodeType*]]

Adds entries to link_dict for Crossrack Transformers.

pyapi_rts.class_extractor.readers package

Subpackages

pyapi_rts.class_extractor.readers.blocks package

Submodules

pyapi_rts.class_extractor.readers.blocks.base_block_reader module

class pyapi_rts.class_extractor.readers.blocks.base_block_reader.**BaseBlockReader**

Bases: `object`

Base class representing an indented block in a file

blocks:
list[*pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*]

A list of blocktypes contained in this block : list[CBlockReader]

line_readers:
list[*pyapi_rts.class_extractor.readers.lines.base_line_reader.BaseLineReader*]

A list of lineReaders searched for in this block : list[CLineReader]

merge_results(*cblock:* pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader) → None

Merge results from another block into this block

Parameters **cblock** (*'CBlockReader'*) – 'CBlockReader'

Returns None

read(*lines: list[str]*) → None

Read a block

Parameters **lines** (*list[str]*) – list of lines in block

Return type None

reg: Pattern

A dictionary containing the results of the block : dict[str, Any]

results: dict[str, Any]

A dictionary containing the results of the block : dict[str, Any]

write_result(*key: str*, *value: Any*)

Appends a result to the results dictionary

Parameters

- **key** (*str*) – Key of the result
- **value** (*Any*) – Value of the result

pyapi_rts.class_extractor.readers.blocks.component_def_file module

class pyapi_rts.class_extractor.readers.blocks.component_def_file.**ComponentDefFile**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads a component definition file

read_from_file(*filename: str*) → bool

Reads the file

Parameters **filename** (*str*) – Path to the file

Returns True if the file was read successfully

Return type bool

pyapi_rts.class_extractor.readers.blocks.computation_transformer module

class pyapi_rts.class_extractor.readers.blocks.computation_transformer.**ComputationTransformer**(**args*, ***kwargs*)

Bases: lark.visitors.Transformer

Transformer for the computation Lark grammar.

acos(*children*)

Transforms a fixed impedance.

addition(*children*)

Transforms an addition.

bcal(*children*)

Transforms a fixed impedance.

boolean_exp(*childern*)

Transforms a boolean expression.

boolean_var(*children*)

Transforms a boolean variable.

calc_l(*children*)

Transforms a calc l.

calc_nm_cond(*children*)

Transforms a number calculation

concat(*childeren*)

Transforms a concat operation.

condition(*children*)

Transforms a condition.

cos(*children*)

Transforms a cosine.

division(*children*)

Transforms a division.

filt_data(*children*)

Transforms a filter data.

fixedimpedance(*children*)

Transforms a fixed impedance.

function_args(*children*)

Transforms a function with one argument.

groupname(*children*)

Transforms a fixed impedance.

hex_to_int(*children*)

Transforms a hexadecimal number to an integer.

internal_function(*children*)

Transforms an internal function.

lcal(*children*)

Transforms a fixed impedance.

lead_lag(*children*)

Transforms a lead-lag.

line(*children*) → tuple[str, type, str]

Transforms the line to Pyhton code.

llcomp(*children*)

Transforms a fixed impedance.

loadf(*children*)

Transforms a fixed impedance.

multiplication(*children*)

Transforms a multiplication.

number(*children*)

Transforms a number.

p_q_calc(*children*)

Transforms a p-q-calc.

p_q_calc_i(*children*)

Transforms a p-q-i-calc.

pcalci(*children*)

Transforms a fixed impedance.

pick_model(*children*)

Transforms a pick_model function.

pick_wye_delta(*children*)

Transforms a pick wye-delta.

pickmodel(*children*)

Transforms a fixed impedance.

picknode(*children*)
Transforms a fixed impedance.

pickv(*children*)
Transforms a fixed impedance.

pickval(*children*)
Transforms a fixed impedance.

pickval2(*children*)
Transforms a fixed impedance.

pickvwgd(*children*)
Transforms a fixed impedance.

pow(*children*)
Transforms a power.

qcalci(*children*)
Transforms a fixed impedance.

rcal(*children*)
Transforms a fixed impedance.

requiv(*children*)
Transforms a requiv.

rnet_calc_pi_yz(*children*)
Transforms a fixed impedance.

shift(*children*)
Transforms a left / right shift.

sin(*children*)
Transforms a sine.

sqrt(*children*)
Transforms a square root.

start(*children*)
Transforms the root of the tree.

statement(*children*)
Transforms a statement.

statement_br(*children*)
Transforms a statement in brackets.

string(*children*)
Transforms a string.

strlen(*children*)
Transforms a string length.

subtraction(*children*)
Transforms a subtraction.

tan(*children*)

Transforms a tangent.

types_lf(*children*)

Transforms a types_lf.

variable(*args*)

Transforms a variable.

xequiv(*children*)

Transforms a xequiv.

pyapi_rts.class_extractor.readers.blocks.computations_block module

class pyapi_rts.class_extractor.readers.blocks.computations_block.**ComputationsBlock**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

A block reader for the computations block.

DOLLAR_WORD_REGEX = re.compile(".*?\\$+([\\w\\.@']+).*")

EURO_WORD_REGEX = re.compile(".*?\\$?€+([\\w\\.@']+).*")

read(*lines: list[str]*) → None

Reads the computations block.

class
pyapi_rts.class_extractor.readers.blocks.computations_block.**LarkComputationSingleton**

Bases: object

SIMPLE_REGEX = re.compile('^\\s*(INTEGER|REAL|STRING)\\s+(\\w+)\\s*=\\s*((?:\\s*
w+\\s*[\\+*\\/\\-]\\s*)*)(\\w+)\\s*$')**

static parse(*computation: str*) → str

Parses a computation.

static tag_values(*string: str*) → str

pyapi_rts.class_extractor.readers.blocks.directives_block module

class pyapi_rts.class_extractor.readers.blocks.directives_block.**DirectivesBlock**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads the DIRECTIVES block from the definition file

class pyapi_rts.class_extractor.readers.blocks.directives_block.**LinkLine**

Bases: *pyapi_rts.class_extractor.readers.lines.base_line_reader.BaseLineReader*

Reads the COMPONENT_LINKED line.

read_line(*line: str*) → bool

Reads a line and extracts information

Parameters **line** (*str*) – Line to read

Returns Success of the read operation

Return type bool

reg: `Pattern = re.compile('LINKED_COMPONENT\\s*=\\s*TRUE.*')`

class pyapi_rts.class_extractor.readers.blocks.directives_block.**NameLine**

Bases: *pyapi_rts.class_extractor.readers.lines.base_line_reader.BaseLineReader*

Reads the name of the parameter naming the component.

read_line(*line: str*) → bool

Reads a line and extracts information

Parameters **line** (*str*) – Line to read

Returns Success of the read operation

Return type bool

reg: `Pattern = re.compile('^NAME = (\\S*)\\n?$')`

class pyapi_rts.class_extractor.readers.blocks.directives_block.**StretchLine**

Bases: *pyapi_rts.class_extractor.readers.lines.base_line_reader.BaseLineReader*

Reads the stretchable line from the definition file and determines if the component is stretchable

read_line(*line: str*) → bool

Determines if the component is stretchable

Parameters **line** (*str*) – Line with the stretchable directive

Raises **ValueError** – Line does not contain a stretchable directive

Returns Success if stretchable directive was found

Return type bool

reg: `Pattern = re.compile('STRETCHABLE = (\\S+).*')`

pyapi_rts.class_extractor.readers.blocks.graphics_block module

class pyapi_rts.class_extractor.readers.blocks.graphics_block.**GraphicsBlock**(*incl_macros: bool = True*)

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads the GRAPHICS block from the definition file.

read(*lines: list[str]*) → None

Read a block

Parameters **lines** (*list[str]*) – list of lines in block

Return type None

pyapi_rts.class_extractor.readers.blocks.node_block module

class pyapi_rts.class_extractor.readers.blocks.node_block.**CompDefNode**(*name: str*, *x: str*, *y: str*, *io:* pyapi_rts.shared.node_type.NodeIO, *_type:* pyapi_rts.shared.node_type.NodeType, *link_name: str*, *phase: str*)

Bases: object

A node read from the node block of a component definition.

as_ext_conn_point() → *pyapi_rts.class_extractor.extracted.ext_connection_point.ExtConnectionPoint*

Returns the node as an ExtConnectionPoint object.

Returns Node converted to an ExtConnectionPoint

Return type *ExtConnectionPoint*

class pyapi_rts.class_extractor.readers.blocks.node_block.**NodeBlock**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads the NODES block from the definition file.

read(*lines: list[str]*) → None

Read a block

Parameters **lines** (*list[str]*) – list of lines in block

Return type None

pyapi_rts.class_extractor.readers.blocks.parameter_block module

class pyapi_rts.class_extractor.readers.blocks.parameter_block.**ParameterBlock**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

A block of parameters.

read(*lines: list[str]*) → None

Reads the parameter block.

Parameters **lines** (*list[str]*) – Lines to read

pyapi_rts.class_extractor.readers.blocks.section_block module

class pyapi_rts.class_extractor.readers.blocks.section_block.**CompDefSection**(*name: str*)

Bases: object

A section of parameters with a name

class pyapi_rts.class_extractor.readers.blocks.section_block.**SectionBlock**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads a section of parameters from the definition file

read(*lines: list[str]*) → None

Reads the section block.

Parameters **lines** (*list[str]*) – Lines to read

Raises **ValueError** – The first line of the section block is not a section name

Module contents

class pyapi_rts.class_extractor.readers.blocks.**BaseBlockReader**

Bases: object

Base class representing an indented block in a file

blocks:
list[*pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*]

A list of blocktypes contained in this block : list[CBlockReader]

line_readers:
list[*pyapi_rts.class_extractor.readers.lines.base_line_reader.BaseLineReader*]

A list of lineReaders searched for in this block : list[CLineReader]

merge_results(*cblock:* pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader) → None

Merge results from another block into this block

Parameters **cblock** (*'CBlockReader'*) – 'CBlockReader'

Returns None

read(*lines: list[str]*) → None

Read a block

Parameters **lines** (*list[str]*) – list of lines in block

Return type None

reg: Pattern

A dictionary containing the results of the block : dict[str, Any]

results: dict[str, Any]

A dictionary containing the results of the block : dict[str, Any]

write_result(*key: str*, *value: Any*)

Appends a result to the results dictionary

Parameters

- **key** (*str*) – Key of the result
- **value** (*Any*) – Value of the result

class pyapi_rts.class_extractor.readers.blocks.**ComponentDefFile**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads a component definition file

read_from_file(*filename: str*) → bool

Reads the file

Parameters **filename** (*str*) – Path to the file

Returns True if the file was read successfully

Return type bool

class pyapi_rts.class_extractor.readers.blocks.**DirectivesBlock**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads the DIRECTIVES block from the definition file

class pyapi_rts.class_extractor.readers.blocks.**GraphicsBlock**(*incl_macros: bool = True*)

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads the GRAPHICS block from the definition file.

read(*lines: list[str]*) → None

Read a block

Parameters **lines** (*list[str]*) – list of lines in block

Return type None

class pyapi_rts.class_extractor.readers.blocks.**NodeBlock**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads the NODES block from the definition file.

read(*lines: list[str]*) → None

Read a block

Parameters **lines** (*list[str]*) – list of lines in block

Return type None

class pyapi_rts.class_extractor.readers.blocks.**ParameterBlock**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

A block of parameters.

read(*lines: list[str]*) → None

Reads the parameter block.

Parameters **lines** (*list[str]*) – Lines to read

class pyapi_rts.class_extractor.readers.blocks.**SectionBlock**

Bases: *pyapi_rts.class_extractor.readers.blocks.base_block_reader.BaseBlockReader*

Reads a section of parameters from the definition file

read(*lines: list[str]*) → None

Reads the section block.

Parameters **lines** (*list[str]*) – Lines to read

Raises **ValueError** – The first line of the section block is not a section name

pyapi_rts.class_extractor.readers.lines package

Submodules

pyapi_rts.class_extractor.readers.lines.base_line_reader module

class pyapi_rts.class_extractor.readers.lines.base_line_reader.**BaseLineReader**

Bases: object

Extracts information from a line matching the given pattern

read_line(*line: str*) → bool

Reads a line and extracts information

Parameters **line** (*str*) – Line to read

Returns Success of the read operation

Return type bool

reg: Pattern = None

results: dict[str, Any]

A dictionary containing the results of the line : dict[str, Any]

return_and_reset() → dict[str, Any]

Returns the results and resets the results dictionary

Returns Results of the read operation

Return type dict[str, Any]

write_result(*key: str*, *value: Any*)

Writes a new entry to the results dictionary at a given key

Parameters

- **key** (*str*) – Key to write to
- **value** (*Any*) – Value of result

pyapi_rts.class_extractor.readers.lines.comp_def_parameter_reader module

class pyapi_rts.class_extractor.readers.lines.comp_def_parameter_reader.**CompDefParameterReader**

Bases: *pyapi_rts.class_extractor.readers.lines.base_line_reader.BaseLineReader*

Reads a parameter line from the definition file

read_line(*line: str*) → None

Extracts information from a line

Parameters **line** (*str*) – Line to read

Raises **ValueError** – Line does not contain a parameter

reg: Pattern = re.compile('(\\S+)\\s+\\"(.*)\\"\\s+\\"([^\\"]*)\\"\\s+(\\S+)\\s+(\\S+)(?:\\s+(\\S+)(?:\\s+(\\S+)(?:\\s+(\\S+)?(?:\\s+(.+)?(?:\\s*))?)?)?)?(?:\\n|$)')

pyapi_rts.class_extractor.readers.lines.condition_line_reader module

class pyapi_rts.class_extractor.readers.lines.condition_line_reader.**ConditionLineReader**

Bases: object

Reads a condition line from the definition file

get_condition(*line: str*) → None | tuple['IfElse', *pyapi_rts.shared.parameter_condition.ParameterCondition*]

Reads the condition from the line without changing the result dictionary.

is_condition_line(*line: str*) → bool

Checks if the line is a condition line or an end line

is_elif_line(*line: str*) → bool

Checks if the line is an elif line

is_else_line(*line: str*) → bool

Checks if the line is an else line

is_end_line(*line: str*) → bool

Checks if the line is an end line

is_if_line(*line: str*) → bool

Checks if the line is a start line

reg = re.compile('')

class pyapi_rts.class_extractor.readers.lines.condition_line_reader.**ConditionLineTree**

Bases: object

class pyapi_rts.class_extractor.readers.lines.condition_line_reader.**IfElse**(*value*)

Bases: enum.Enum

Enum for the different types of condition

Parameters **Enum** (*Enum*) – IF,ELSE,ELSE

ELIF = 1

ELSE = 2

IF = 0

IFNOT = 3

pyapi_rts.class_extractor.readers.lines.graphics_condition_line_reader module

class pyapi_rts.class_extractor.readers.lines.graphics_condition_line_reader.**GraphicsConditionLineReade**

Bases: *pyapi_rts.class_extractor.readers.lines.condition_line_reader.ConditionLineReader*

Reads condition lines from the GRAPHICS: block of the Component Definition Files.

get_condition(*line: str*) → None | tuple[*pyapi_rts.class_extractor.readers.lines.condition_line_reader.IfElse*, *pyapi_rts.shared.parameter_condition.ParameterCondition*]

Reads the condition from the line without changing the result dictionary.

get_line_components(*line: str*) → list[str]

is_elif_line(*line: str*) → bool

Checks if the line is an elif line

is_else_line(*line: str*) → bool

Checks if the line is an else line

is_end_line(*line: str*) → bool

Checks if the line is an end line

is_if_line(*line: str*) → bool

Checks if the line is a start line

reg = re.compile('\\s*(?:(IfNot)|(ElseIf)|(If)|(Else))(?:)?\\(?(.*)\\)?', re.IGNORECASE)

pyapi_rts.class_extractor.readers.lines.node_condition_line_reader module

class pyapi_rts.class_extractor.readers.lines.node_condition_line_reader.**NodeConditionLineReader**

Bases: *pyapi_rts.class_extractor.readers.lines.condition_line_reader.ConditionLineReader*

Reads condition lines from the NODES: block of the Component Definition Files.

get_condition(*line: str*) → None | tuple['IfElse', *pyapi_rts.shared.parameter_condition.ParameterCondition*]

Reads the condition from the line without changing the result dictionary.

is_elif_line(*line: str*) → bool

Checks if the line is an elif line

is_else_line(*line: str*) → bool

Checks if the line is an else line

is_end_line(*line: str*) → bool

Checks if the line is an end line

is_if_line(*line: str*) → bool

Checks if the line is a start line

reg = re.compile('.*#(?:(ElseIf|ELseIf|ELSEIF)|(ELSE|Else)|(IF|If))(?:)?(\\(?.*\\)?)')

Module contents

class pyapi_rts.class_extractor.readers.lines.**BaseLineReader**

Bases: object

Extracts information from a line matching the given pattern

read_line(*line: str*) → bool

Reads a line and extracts information

Parameters **line** (*str*) – Line to read

Returns Success of the read operation

Return type bool

reg: Pattern = None

results: dict[str, Any]

A dictionary containing the results of the line : dict[str, Any]

return_and_reset() → dict[str, Any]

Returns the results and resets the results dictionary

Returns Results of the read operation

Return type dict[str, Any]

write_result(*key: str*, *value: Any*)

Writes a new entry to the results dictionary at a given key

Parameters

- **key** (*str*) – Key to write to
- **value** (*Any*) – Value of result

class pyapi_rts.class_extractor.readers.lines.**CompDefParameterReader**

Bases: *pyapi_rts.class_extractor.readers.lines.base_line_reader.BaseLineReader*

Reads a parameter line from the definition file

read_line(*line: str*) → None

Extracts information from a line

Parameters **line** (*str*) – Line to read

Raises **ValueError** – Line does not contain a parameter

reg: Pattern = re.compile('(\\S+)\\s+\\"(.*)\\"\\s+\\"([^\\"]*)\\"\\s+(\\S+)\\s+(\\S+)(?:\\s+(\\S+)(?:\\s+(\\S+)(?:\\s+(\\S+)?(?:\\s+(.+)?(?:\\s*))?)?)?)?(?:\\n|$)')

results: dict[str, Any]

A dictionary containing the results of the line : dict[str, Any]

class pyapi_rts.class_extractor.readers.lines.**ConditionLineReader**

Bases: object

Reads a condition line from the definition file

get_condition(*line: str*) → None | tuple['IfElse', *pyapi_rts.shared.parameter_condition.ParameterCondition*]

Reads the condition from the line without changing the result dictionary.

is_condition_line(*line: str*) → bool

Checks if the line is a condition line or an end line

is_elif_line(*line: str*) → bool

Checks if the line is an elif line

is_else_line(*line: str*) → bool

Checks if the line is an else line

is_end_line(*line: str*) → bool

Checks if the line is an end line

is_if_line(*line: str*) → bool

Checks if the line is a start line

reg = re.compile('')

class pyapi_rts.class_extractor.readers.lines.**GraphicsConditionLineReader**(*incl_macros: bool = True*)

Bases: *pyapi_rts.class_extractor.readers.lines.condition_line_reader.ConditionLineReader*

Reads condition lines from the GRAPHICS: block of the Component Definition Files.

get_condition(*line: str*) → None | tuple[*pyapi_rts.class_extractor.readers.lines.condition_line_reader.IfElse*, *pyapi_rts.shared.parameter_condition.ParameterCondition*]

Reads the condition from the line without changing the result dictionary.

get_line_components(*line: str*) → list[str]

is_elif_line(*line: str*) → bool

Checks if the line is an elif line

is_else_line(*line: str*) → bool

Checks if the line is an else line

is_end_line(*line: str*) → bool

Checks if the line is an end line

is_if_line(*line: str*) → bool

Checks if the line is a start line

reg = re.compile('\\s*(?:(IfNot)|(ElseIf)|(If)|(Else))(?:)?\\(?(.*)\\)?', re.IGNORECASE)

class pyapi_rts.class_extractor.readers.lines.**NodeConditionLineReader**

Bases: *pyapi_rts.class_extractor.readers.lines.condition_line_reader.ConditionLineReader*

Reads condition lines from the NODES: block of the Component Definition Files.

get_condition(*line: str*) → None | tuple['IfElse', *pyapi_rts.shared.parameter_condition.ParameterCondition*]

Reads the condition from the line without changing the result dictionary.

is_elif_line(*line: str*) → bool

Checks if the line is an elif line

is_else_line(*line: str*) → bool

Checks if the line is an else line

is_end_line(*line: str*) → bool

Checks if the line is an end line

is_if_line(*line: str*) → bool

Checks if the line is a start line

reg = re.compile('.*#(?:(ElseIf|ELseIf|ELSEIF)|(ELSE|Else)|(IF|If))(?:)?(\\(?.*\\)?)')

Module contents

Submodules

pyapi_rts.class_extractor.enum_hash_pool module

class pyapi_rts.class_extractor.enum_hash_pool.**EnumHashPool**

Bases: `object`

Manages a collection of ExtEnumParameters in a hash table.

add(*component:* pyapi_rts.class_extractor.extracted.ext_component.ExtComponent, *enum:* pyapi_rts.class_extractor.extracted.ext_enum_parameter.ExtEnumParameter)

Adds an ExtEnumParameter to the hash table.

Parameters **enum** (`ExtEnumParameter`) – The Enum Parameter to add.

get_hash(*name: str*) → int

Returns the hash of the enum parameter with the given name.

load_from_file(*pool_path: str*) → bool

Load the enum pool from a file and generate the enum hash pool.

Parameters **path** (*str*) – The path to the file in enum pool format.

Returns list of enum parameters.

Return type list[*ExtEnumParameter*]

property pool

Returns the pool.

remove_tailing_digits(*string: str*) → str

Removes the trailing digits from a string.

Parameters **s** (*str*) – The string to remove the trailing digits from.

Returns The string without the trailing digits.

Return type str

pyapi_rts.class_extractor.graphics_parsing module

pyapi_rts.class_extractor.graphics_parsing.**box_to_coord**(*groups: tuple*, *norotate: bool*, *nomirror: bool*) → Union[tuple, *pyapi_rts.shared.bounding_box.BoundingBox*]

pyapi_rts.class_extractor.graphics_parsing.**circle_to_coord**(*groups: tuple*, *norotate: bool*, *nomirror: bool*) → Union[tuple, *pyapi_rts.shared.bounding_box.BoundingBox*]

pyapi_rts.class_extractor.graphics_parsing.**line_to_coord**(*groups: tuple*, *norotate: bool*, *nomirror: bool*) → Union[tuple, *pyapi_rts.shared.bounding_box.BoundingBox*]

pyapi_rts.class_extractor.graphics_parsing.**text_to_coord**(*groups: tuple*, *norotate: bool*, *nomirror: bool*) → Union[tuple, *pyapi_rts.shared.bounding_box.BoundingBox*]

pyapi_rts.class_extractor.graphics_parsing.**to_to_coord**(*groups: tuple*, *norotate: bool*, *nomirror: bool*) → Union[tuple, *pyapi_rts.shared.bounding_box.BoundingBox*]

pyapi_rts.class_extractor.main module

pyapi_rts.class_extractor.main.**read_component_dir**(*dir_path: str*, *tag_dict: dict[str, list[str]]*, *include_obsolete: bool = False*, *worker_count: int = 8*) → list[tuple[*pyapi_rts.class_extractor.extracted.ext_component.ExtComp*
list[*pyapi_rts.class_extractor.extracted.ext_enum_parameter.ExtEnum*

Reads the contents of a directory.

Parameters

- **path** (*str*) – The path to the directory.
- **tag_dict** (*dict[str, list[str]]*) – Dictionary with component tags (read_component_tags()).
- **include_obsolete** (*bool*) – Include obsolete components.
- **worker_count** (*int*) – The number of workers/threads to use.

Returns list of components and their enum parameter types.

Return type list[tuple[*ExtComponent*, list[*ExtEnumParameter*]]]

pyapi_rts.class_extractor.main.**read_component_tags**(*path: str*) → dict[str, list[str]]

Reads the component tags from a file.

Parameters **path** (*str*) – Path to the file.

Returns Dictionary with component tags (Component Type Name -> Tag list).

Return type dict[str, list[str]]

pyapi_rts.class_extractor.main.**read_file**(*file_path: str*, *tag_dict: dict[str, list[str]]*) → tuple[*pyapi_rts.class_extractor.extracted.ext_component.ExtComponent*, list[*pyapi_rts.class_extractor.extracted.ext_enum_parameter.ExtEnumParameter*]

Reads a component definition file.

Parameters

- **path** (*str*) – Path to a component definition file.
- **tag_dict** (*dict[str, list[str]]*) – Dictionary with component tags (read_component_tags()).

Returns The component and the list of enum parameters.(None, []) if the file could not be read.

Return type tuple[*ExtComponent*, list[*ExtEnumParameter*]]

pyapi_rts.class_extractor.main.**read_graphics_files**(*paths*)

pyapi_rts.class_extractor.main.**reverse_dictionary**(*dictionary: dict[Any, list[Any]]*) → dict

Reverses a dictionary with multiple entries per key.

Parameters **dictionary** (*dict[Any, list[Any]]*) – Dictionary to reverse.

Returns Reversed dictionary.

Return type dict

pyapi_rts.class_extractor.utils module

pyapi_rts.class_extractor.utils.**valid_file_name**(*string: str*) → str

Converts a string to a valid file name

Parameters **string** (*str*) – The string to convert

Returns The converted string

Return type str

Module contents

ClassExtractor converts a folder of Component Builder files to Python classses representing the components.

class pyapi_rts.class_extractor.**EnumHashPool**

Bases: object

Manages a collection of ExtEnumParameters in a hash table.

add(*component:* pyapi_rts.class_extractor.extracted.ext_component.ExtComponent, *enum:* pyapi_rts.class_extractor.extracted.ext_enum_parameter.ExtEnumParameter)

Adds an ExtEnumParameter to the hash table.

Parameters **enum** (ExtEnumParameter) – The Enum Parameter to add.

get_hash(*name: str*) → int

Returns the hash of the enum parameter with the given name.

load_from_file(*pool_path: str*) → bool

Load the enum pool from a file and generate the enum hash pool.

Parameters **path** (*str*) – The path to the file in enum pool format.

Returns list of enum parameters.

Return type list[*ExtEnumParameter*]

property pool

Returns the pool.

remove_tailing_digits(*string: str*) → str

Removes the trailing digits from a string.

Parameters **s** (*str*) – The string to remove the trailing digits from.

Returns The string without the trailing digits.

Return type str

pyapi_rts.shared package

Submodules

pyapi_rts.shared.bounding_box module

class pyapi_rts.shared.bounding_box.**BoundingBox**(*x1: int | str*, *y1: int | str*, *x2: int | str*, *y2: int | str*, *norotate: bool = False*, *nomirror: bool = False*)

Bases: object

The bounding box of a component rectangle.

evaluate(*dictionary*, *rotation=0*, *mirror=0*) → tuple[int, int, int, int]

Evaluates the parameter bound bounding box to an integer tuple. :return: The integer tuple. :rtype: tuple[int, int, int, int]

init_code()

pyapi_rts.shared.component_hook module

class pyapi_rts.shared.component_hook.**ComponentHook**

Bases: object

Base class for components to be hooked into the main program.

classmethod graph_connections(*components*, *pos_dict: dict*, *link_dict: dict*) → list[tuple[str, str, str]]

Hook method.

classmethod link_connections(*components: list*) → list[tuple[str, str, str, *pyapi_rts.shared.node_type.NodeType*]]

Hook for adding entries to link_dict. :param components: list of components :type components: list[Component] :return: list of connections in form [(name, component_uuid, point_name, node_type), ...] :rtype: list[tuple[str, str, node_type]]

classmethod special_value(*component: Any*, *key: str*) → Optional[Any]

Adds new special values to components. :param component: Component to evaluate. :type component: Component :return: Value of the special key or None if it does not exist for this component. :rtype: Any | None

pyapi_rts.shared.condition_tree module

class pyapi_rts.shared.condition_tree.**BBNode**

Bases: *pyapi_rts.shared.condition_tree.ConditionTreeNode*

to_code() → list[str]

class pyapi_rts.shared.condition_tree.**CPNode**

Bases: *pyapi_rts.shared.condition_tree.ConditionTreeNode*

to_code() → list[str]

class pyapi_rts.shared.condition_tree.**ConditionTreeNode**

Bases: `object`

A generic class for nodes in a condition tree.

to_code() → list[str]

class pyapi_rts.shared.condition_tree.**IfNode**(*condition*)

Bases: *pyapi_rts.shared.condition_tree.ConditionTreeNode*

A condition tree node that has condition and contains a list of other nodes.

body: **list[***pyapi_rts.shared.condition_tree.ConditionTreeNode***]**

The list of nodes contained in this node.

condition: *pyapi_rts.shared.parameter_condition.ParameterCondition*

The condition of the node.

to_code() → list[str]

class pyapi_rts.shared.condition_tree.**NewConditionTree**(*if_branch*)

Bases: *pyapi_rts.shared.condition_tree.ConditionTreeNode*

A condition tree that contains an if branch, and optionally an else branch and multiple elif branches.

elif_branches: **list[***pyapi_rts.shared.condition_tree.IfNode***]**

The optional elif branches, consisting of a list of if nodes.

else_branch: **list[***pyapi_rts.shared.condition_tree.ConditionTreeNode***]**

The optional else branch, consisting of a list of condition tree nodes.

if_branch: *pyapi_rts.shared.condition_tree.IfNode*

The mandatory if branch of the tree, consisting of a single IfNode.

to_code() → list[str]

pyapi_rts.shared.node_type module

class pyapi_rts.shared.node_type.**NodeIO**(*value*)

Bases: `enum.Enum`

Enum for the different types of nodes

Parameters **Enum** (*Enum*) – INPUT, OUTPUT, IO, EXTERNAL, UNDEFINED, DEFAULT, GROUND, SHORT, FPGA_SOLVER, VSC, ELECTRICAL

DEFAULT = 'DEFAULT'

ELECTRICAL = 'ELECTRICAL'

EXTERNAL = 'EXTERNAL'

FPGA_SOLVER = 'FPGA_SOLVER'

GROUND = 'GROUND'

INPUT = 'INPUT'

IO = 'I/O'

OUTPUT = 'OUTPUT'

SHORT = 'SHORT'

UNDEFINED = 'UNDEFINED'

VSC = 'VSC'

class pyapi_rts.shared.node_type.**NodeType**(*value*)

Bases: enum.Enum

Enum for the different types of nodes

Parameters **Enum** (*Enum*) – NC_CONNECTED_LINKED, NC_LINKED, OTHER

NC_CONNECTED_LINKED = 'NAME_CONNECTED:LINKED'

NC_LINKED = 'NAME_CONNECTED'

OTHER = 'OTHER'

pyapi_rts.shared.parameter_bound_property module

class pyapi_rts.shared.parameter_bound_property.**ParameterBoundProperty**(*value: Union[Any, str]*, *_type: type*)

Bases: object

A property that can be bound to a parameter or an explicit value.

INNER_BRACKET_PATTERN = re.compile('\\$\\((.*)\\((.*)\\)(.*)\\)')

MULTIPLICATION_PATTERN = re.compile('\\$\\((.*\\s)?(-?[A-z_\\d\\.]+)\\s*(*)\\s*(-?[A-z_\\d\\.]+)(.*)\\)')

OPERATOR_PATT = re.compile('\\$\\((-?.*?)\\s*([+%-])\\s*(-?[A-z_\\d\\.]+)\\s*\\)')

SINGLE_VALUE_BRACKETS_PATTERN = re.compile('\\$\\((\\s*[A-z_\\-\\d\\.]+)\\s*\\)')

get_direct_value() → Any

Returns the value of the parameter bound property.

Returns The value of the property

Return type Any

get_value(*dictionary: Optional[dict] = None*) → Union[Any, str]

Returns the value of the parameter bound property.

Parameters **dictionary** (*dict, optional*) – The dictionary of a component's parameters

Returns The value of the property

Return type Any | str

set_value(*value: Union[Any, str]*)

Sets the value of the parameter bound property.

Parameters **value** (*Any | str*) – The value of the property

pyapi_rts.shared.parameter_condition module

class pyapi_rts.shared.parameter_condition.**OperatorChainOperator**(*value*)

Bases: enum.Enum

Enum of all possible operator chain operators. Composed of the check function and the string representation of the operator.

AND = (<function OperatorChainOperator.<lambda>>, '&')

AND2 = (<function OperatorChainOperator.<lambda>>, '&&')

LEFT = (<function OperatorChainOperator.<lambda>>, '\n')

OR = (<function OperatorChainOperator.<lambda>>, '|')

OR2 = (<function OperatorChainOperator.<lambda>>, '||')

class pyapi_rts.shared.parameter_condition.**ParameterCondition**(*left: Union[*pyapi_rts.shared.parameter_bound_property.Pa pyapi_rts.shared.parameter_condition.ParameterCondi *right: Union[*pyapi_rts.shared.parameter_bound_property.Pa pyapi_rts.shared.parameter_condition.ParameterCondi *operator: Union[*pyapi_rts.shared.parameter_condition.Paramete pyapi_rts.shared.parameter_condition.OperatorChainO *negate: bool = False*)

Bases: object

A condition that compares two ParameterBoundProperty objects

check(*dictionary*) → bool

Evaluates the condition on a dictionary of a component's parameters

Parameters **dictionary** (*dict[str, Any]*) – The dictionary of parameters to evaluate the condition on

Returns True if the condition is met, False if not

Return type bool

classmethod empty()

Returns an empty ParameterCondition that always returns True

Returns An empty ParameterCondition

Return type _type_

left: *pyapi_rts.shared.parameter_bound_property.ParameterBoundProperty* | *pyapi_rts.shared.parameter_condition.ParameterCondition*

The left side of the condition

negate

If True, negate the evaluation of the condition.

operator: *pyapi_rts.shared.parameter_condition.ParameterConditionOperator* | *pyapi_rts.shared.parameter_condition.OperatorChainOperator*

The operator of the condition

right: *pyapi_rts.shared.parameter_bound_property.ParameterBoundProperty* | *pyapi_rts.shared.parameter_condition.ParameterCondition*

The right side of the condition

classmethod single(*lst: list[Any]*)

Returns a parameter condition that always returns the node_list

Parameters **node_list** (*list[Any]*) – The node list to always return

Returns A parameter condition that always returns the node_list

Return type tuple[*ParameterCondition*, list[Any]]

class pyapi_rts.shared.parameter_condition.**ParameterConditionOperator**(*value*)

Bases: enum.Enum

Enum of all possible parameter condition operators. Composed of a function that evaluates the condition and a string representation of the operator

EQUAL = (<function ParameterConditionOperator.<lambda>>, '=')

EQUAL2 = (<function ParameterConditionOperator.<lambda>>, ',')

GREATER_THAN = (<function ParameterConditionOperator.<lambda>>, '>')

GREATER_THAN_OR_EQUAL = (<function ParameterConditionOperator.<lambda>>, '>=')

LESS_THAN = (<function ParameterConditionOperator.<lambda>>, '<')

LESS_THAN_OR_EQUAL = (<function ParameterConditionOperator.<lambda>>, '<=')

NONE = (<function ParameterConditionOperator.<lambda>>, '\n')

NOT_EQUAL = (<function ParameterConditionOperator.<lambda>>, '!=')

TOGGLE_EQUAL = (<function ParameterConditionOperator.<lambda>>, '==')

pyapi_rts.shared.parameter_condition.**get_enum_index**(*enum_value: Any*) → int

Returns the index of an enum value

Parameters **enumValue** (*Any*) – The enum value to get the index of

Returns The index of the enum value

Return type int

pyapi_rts.shared.parameter_condition.**get_with_enum_as_index**(*value: Any*) → Any

Returns the index of an enum value if it is an enum value, otherwise returns the value

Parameters **value** (*Any*) – The value to get the index of

Returns The index of the enum value if it is an enum value, otherwise returns the value

Return type Any

pyapi_rts.shared.stretchable module

class pyapi_rts.shared.stretchable.**Stretchable**(*value*)

Bases: enum.Enum

Enum for the stretchable directives

BOX = ('STRETCHABLE_BOX',)

NO = ('NO',)

UP_DOWN = ('STRETCHABLE_UP_DOWN_LINE',)

Module contents

Shared classes between the modules of pyapi_rts.

class pyapi_rts.shared.**BoundingBox**(*x1: int | str*, *y1: int | str*, *x2: int | str*, *y2: int | str*, *norotate: bool = False*, *nomirror: bool = False*)

Bases: object

The bounding box of a component rectangle.

evaluate(*dictionary*, *rotation=0*, *mirror=0*) → tuple[int, int, int, int]

Evaluates the parameter bound bounding box to an integer tuple. :return: The integer tuple. :rtype: tuple[int, int, int, int]

init_code()

class pyapi_rts.shared.**ComponentHook**

Bases: object

Base class for components to be hooked into the main program.

classmethod graph_connections(*components*, *pos_dict: dict*, *link_dict: dict*) → list[tuple[str, str, str]]

Hook method.

classmethod link_connections(*components: list*) → list[tuple[str, str, str, *pyapi_rts.shared.node_type.NodeType*]]

Hook for adding entries to link_dict. :param components: list of components :type components: list[Component] :return: list of connections in form [(name, component_uuid, point_name, node_type), ...] :rtype: list[tuple[str, str, node_type]]

classmethod special_value(*component: Any*, *key: str*) → Optional[Any]

Adds new special values to components. :param component: Component to evaluate. :type component: Component :return: Value of the special key or None if it does not exist for this component. :rtype: Any | None

class pyapi_rts.shared.**NodeIO**(*value*)

Bases: enum.Enum

Enum for the different types of nodes

Parameters **Enum** (*Enum*) – INPUT, OUTPUT, IO, EXTERNAL, UNDEFINED, DEFAULT, GROUND, SHORT, FPGA_SOLVER, VSC, ELECTRICAL

DEFAULT = 'DEFAULT'

ELECTRICAL = 'ELECTRICAL'

EXTERNAL = 'EXTERNAL'

FPGA_SOLVER = 'FPGA_SOLVER'

GROUND = 'GROUND'

INPUT = 'INPUT'

IO = 'I/O'

OUTPUT = 'OUTPUT'

SHORT = 'SHORT'

UNDEFINED = 'UNDEFINED'

VSC = 'VSC'

class pyapi_rts.shared.**NodeType**(*value*)

Bases: enum.Enum

Enum for the different types of nodes

Parameters **Enum** (*Enum*) – NC_CONNECTED_LINKED, NC_LINKED, OTHER

NC_CONNECTED_LINKED = 'NAME_CONNECTED:LINKED'

NC_LINKED = 'NAME_CONNECTED'

OTHER = 'OTHER'

class pyapi_rts.shared.**OperatorChainOperator**(*value*)

Bases: enum.Enum

Enum of all possible operator chain operators. Composed of the check function and the string representation of the operator.

AND = (<function OperatorChainOperator.<lambda>>, '&')

AND2 = (<function OperatorChainOperator.<lambda>>, '&&')

LEFT = (<function OperatorChainOperator.<lambda>>, '\n')

OR = (<function OperatorChainOperator.<lambda>>, '|')

OR2 = (<function OperatorChainOperator.<lambda>>, '||')

class pyapi_rts.shared.**ParameterBoundProperty**(*value: Union[Any, str]*, *_type: type*)

Bases: object

A property that can be bound to a parameter or an explicit value.

INNER_BRACKET_PATTERN = re.compile('\\$\\((.*)\\((.*)\\)(.*)\\)')

MULTIPLICATION_PATTERN = re.compile('\\$\\((.*\\s)?(-?[A-z_\\d\\.]+)\\s*(*)\\s*(-?[A-z_\\d\\.]+)(.*)\\)')

OPERATOR_PATT = re.compile('\\$\\((-?.*?)\\s*([+%-])\\s*(-?[A-z_\\d\\.]+)\\s*\\)')

SINGLE_VALUE_BRACKETS_PATTERN = re.compile('\\$\\((\\s*[A-z_\\-\\d\\.]+)\\s*\\)')

get_direct_value() → Any

Returns the value of the parameter bound property.

Returns The value of the property

Return type Any

get_value(*dictionary: Optional[dict] = None*) → Union[Any, str]

Returns the value of the parameter bound property.

Parameters **dictionary** (*dict, optional*) – The dictionary of a component's parameters

Returns The value of the property

Return type Any | str

set_value(*value: Union[Any, str]*)

Sets the value of the parameter bound property.

Parameters **value** (*Any | str*) – The value of the property

class pyapi_rts.shared.**ParameterCondition**(*left: Union[*pyapi_rts.shared.parameter_bound_property.ParameterBoundProperty, pyapi_rts.shared.parameter_condition.ParameterCondition*]*, *right: Union[*pyapi_rts.shared.parameter_bound_property.ParameterBoundProperty, pyapi_rts.shared.parameter_condition.ParameterCondition*]*, *operator: Union[*pyapi_rts.shared.parameter_condition.ParameterConditionOperator, pyapi_rts.shared.parameter_condition.OperatorChainOperator*]*, *negate: bool = False*)

Bases: object

A condition that compares two ParameterBoundProperty objects

check(*dictionary*) → bool

Evaluates the condition on a dictionary of a component's parameters

Parameters **dictionary** (*dict[str, Any]*) – The dictionary of parameters to evaluate the condition on

Returns True if the condition is met, False if not

Return type bool

classmethod empty()

Returns an empty ParameterCondition that always returns True

Returns An empty ParameterCondition

Return type _type_

left: *pyapi_rts.shared.parameter_bound_property.ParameterBoundProperty* | *pyapi_rts.shared.parameter_condition.ParameterCondition*

The left side of the condition

negate

If True, negate the evaluation of the condition.

operator: *pyapi_rts.shared.parameter_condition.ParameterConditionOperator* | *pyapi_rts.shared.parameter_condition.OperatorChainOperator*

The operator of the condition

right: *pyapi_rts.shared.parameter_bound_property.ParameterBoundProperty* | *pyapi_rts.shared.parameter_condition.ParameterCondition*

The right side of the condition

classmethod single(*lst: list[Any]*)

Returns a parameter condition that always returns the node_list

Parameters **node_list** (*list[Any]*) – The node list to always return

Returns A parameter condition that always returns the node_list

Return type tuple[*ParameterCondition*, list[Any]]

class pyapi_rts.shared.**ParameterConditionOperator**(*value*)

Bases: enum.Enum

Enum of all possible parameter condition operators. Composed of a function that evaluates the condition and a string representation of the operator

EQUAL = (<function ParameterConditionOperator.<lambda>>, '=')

EQUAL2 = (<function ParameterConditionOperator.<lambda>>, ',')

GREATER_THAN = (<function ParameterConditionOperator.<lambda>>, '>')

GREATER_THAN_OR_EQUAL = (<function ParameterConditionOperator.<lambda>>, '>=')

LESS_THAN = (<function ParameterConditionOperator.<lambda>>, '<')

LESS_THAN_OR_EQUAL = (<function ParameterConditionOperator.<lambda>>, '<=')

NONE = (<function ParameterConditionOperator.<lambda>>, '\n')

NOT_EQUAL = (<function ParameterConditionOperator.<lambda>>, '!=')

TOGGLE_EQUAL = (<function ParameterConditionOperator.<lambda>>, '==')

class pyapi_rts.shared.**Stretchable**(*value*)

Bases: enum.Enum

Enum for the stretchable directives

BOX = ('STRETCHABLE_BOX',)

```
NO = ('NO',)

UP_DOWN = ('STRETCHABLE_UP_DOWN_LINE',)
```

6.1.2 Module contents

pyapi_rts: A Python API to create and modify RSCAD files.

CHAPTER

SEVEN

CLASS EXTRACTOR USAGE

The *ClassExtractor* tool is a code generator for the classes representing the RSCAD components and dependent classes. It is (almost) idempotent, so its output is reproducible over every supported platform.

Attention: The ClassExtractor needs to be run before the first use of pyapi_rts.

In addition, it should be used every time the RSCAD FX version changes or new user defined components are added.

7.1 Requirements

- Python >= 3.10
- RSCAD FX >= 1.0 installed **OR** the COMPONENTS folder from a RSCAD FX installation

7.2 Basic usage

1. Copy the *RSCAD FX x.x/MLIB/COMPONENTS* directory (Windows: *C://Program Files/RTDS/RSCAD FX x.x/...*) to the pyapi_rts/class_extractor/COMPONENTS directory.

OR

Use the –path option to specify the path to the RSCAD FX components directory.

2. **Run the ClassExtractor tool:**

```
>>> poetry run python ./pyapi_rts/class_extractor/main.py
```

Options:

- -h / –help: show the help message and exit
- –delete / -d : delete any previously generated classes
- –path / -p : specify the path to the RSCAD FX components directory
- –includeobsolete / -i : include components in the OBSOLETE folder
- –threads / -t : specify the number of threads to use (default: 8)

7.3 Files used by the ClassExtractor

7.3.1 ComponentBuilder files directory

The COMPONENTS folder from your RSCAD FX installation.

7.3.2 Extensions directory

(see *Extensions*) The directory to place the extensions (directories) in.

The requirements for those directories are defined on the *Extensions* page.

7.3.3 Hooks directory

(see *Hooks*) The directory to place the hooks (Python classes implementing `ComponentHook`) in.

7.3.4 Component Tags (component_tags.txt)

A list of tags for components. This can provide additional information for components that is not contained in the Component Builder file.

The format is a list of tags with the tag name and the components it applies to indented in the following lines, one component per line.

Currently supported tags are:

Tag	Description
connecting	The component is used for connecting other components. Examples are wires, buses and similar components.
hierarchy_connecting	
label	The component can label a bus or other connection.

7.3.5 Initial Enum Pool (enum_pool.txt)

The initial Enum Pool used during component generation. The enum pool is a set of enumerations with distinct options from each other. By pre-defining some enums in this set, it is possible to assign certain names to common enums. The enum values are case-sensitive.

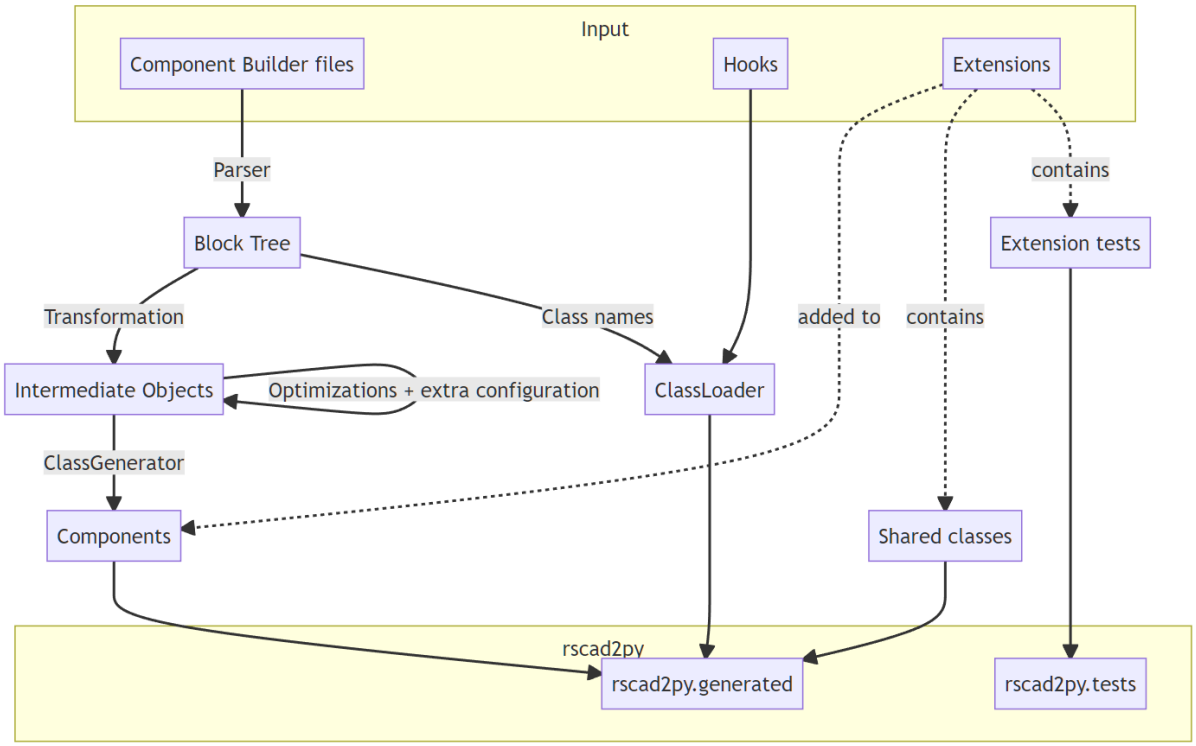
```
ENUM
<name>
<value1>
<value2>
```

(continues on next page)

(continued from previous page)

```
...
END
...
```

7.4 ClassExtractor Structure



Because all of the inputs are considered a part of the `class_extractor` module and the `pyapi_rts.generated` module is derived from it, the 'generated' module is used by the unit tests and generated as part of the testing pipeline.

The exception to this are Extensions, as their structure makes them likely to break after changes to the `class_extractor` module. Extensions are only includes in the component generation in a second run, after a run of the ClassExtractor without extensions resulted in passing tests.

7.5 Outputs

7.5.1 Components

Representations of the component types included in RSCAD FX.

7.5.2 Enums

Enums used by parameters of RSCAD components, shared between components to save memory and storage.

7.5.3 class_loader.py

Includes a set of attributes and methods used to lazy-load the generated classes and execute the hooks.

1. `get_by_key(key: str) -> Component`: Loads and caches the Component class and returns a new instance.
2. `hooks() -> list[ComponentHook]`: Returns a list of all the hooks.

CHAPTER

EIGHT

EXAMPLES

8.1 Creation of empty model

Listing 1: Create and save empty model

```
from pyapi_rts.api.draft import Draft
from pyapi_rts.api.subsystem import Subsystem

if __name__ == '__main__':
    draft = Draft()
    subsystem = Subsystem(draft, 1)
    subsystem.canvas_size_x = 1000
    subsystem.canvas_size_y = 1000
    subsystem.tab_name = "Test"
    draft.add_subsystem(subsystem)
    draft.write_file("test.dfx")
```

8.2 Basic editing of model

Listing 2: Simple example

```
from pyapi_rts.api import *

# Load a RSCAD file
draft = Draft()
draft.read_file(PATH / "bus_rings.dfx")

# Get a specific component and the components connected to it
buslabel1 = draft.subsystems[0].search_by_name("BUS1")[0]
connected_to_bus1 = draft.subsystems[0].get_connected_to(buslabel1)
```

Listing 3: Load, edit and save model

```
import pathlib
from pyapi_rts.api.draft import Draft
from pyapi_rts.api.subsystem import Subsystem
from pyapi_rts.classext2.generated.BUSComponent.BUSComponent import BUSComponent

```

(continues on next page)

(continued from previous page)

```
PATH = pathlib.Path(__file__).parent.resolve()

if __name__ == '__main__':
    draft = Draft()
    draft.read_file(PATH / 'test.dfx')
    bus_component : BUSComponent = draft.get_components()[0]
    bus_component.BUSComponent__CONFIGURATION.SCOL.set_str('RED')
    draft.subsystems[0].modify_component(bus_component)
    draft.write_file(PATH / 'test_out.dfx')
```

Listing 4: From the thesis, p. 39 (Bus coloring)

```
import networkx as nx
import random

from pyapi_rts.api import *
from pyapi_rts.generated.BUSComponent import BUSComponent
from pyapi_rts.generated.rtdssharcsldBUSLABELComponent import␣
↪rtdssharcsldBUSLABELComponent
from pyapi_rts.generated.enums.ScolEnumParameter import ScolEnum

fx = Draft()
fx.read_file("ieee14.dfx")

G = fx._subsystems[0].get_connection_graph()
SG = G.subgraph([n for n, attrdict in G.nodes(data=True) if not "3P2W" in attrdict['type
↪']])

start_nodes = [n for n, ad in G.nodes(data=True) if ad['type'] == 'rtds_sharc_sld_
↪BUSLABEL']
bus_list = fx.get_components_by_type("BUS", False)

colors = list(ScolEnum)
colors.remove(ScolEnum.WHITEWHITE)

for start in start_nodes:
    id_list = list(nx.dfs_postorder_nodes(SG, source=start))
    col = random.choice(colors)
    count = 0
    for bus in bus_list:
        if bus.uuid in id_list:
            bus: BUSComponent = bus
            bus.CONFIGURATION.SCOL.set_value(col)
            count += 1
            fx._subsystems[0].modify_component(bus)
    buslabel: rtdssharcsldBUSLABELComponent = fx._subsystems[0].get_by_id(
        start, False)
    buslabel.Parameters.COL.set_value(col)
    print(f"Found {count} buses, coloring with {col}")
    fx._subsystems[0].modify_component(buslabel)

fx.write_file("ieee_out.dfx")
```

CHAPTER

NINE

.DFX FILE FORMAT

9.1 Introduction

The .dfx file format is a format used by RSCAD FX 1.0 and later versions to store an energy network. It is a plain-text based format that used indentations to define the structure of the file as a tree, with key-value pairs at the leaves.

The nodes of this tree come in two formats, **Type A** and **Type B**. **Type A** nodes consist of a title line ending in a colon, followed by the indented content of the node. **Type B** nodes start and end with a line '{title}-START:' and '{title}-END:' respectively. The content of the node is not indented.

Listing 1: Example of a Type A node.

```
GRAPHICS:
    CANVAS_WIDTH: 1481
    CANVAS_HEIGHT: 568
    CURRENT_SUBSYSTEM_IDX: 0
    DEFAULT_VIEW_MODE: 3
    DEFAULT_ZOOM: 100
    DEFAULT_TOP_LEFT_POINT: 0,0
```

Listing 2: Example of a Type B node.

```
PARAMETERS-START:
LW1: 0.5
SCOL: ORANGE
PARAMETERS-END:
```

9.1.1 Structure of the .dfx file

A .dfx file consists of multiple sections making up the tree.

1. The first line, starting with 'DRAFT', followed by the format version.
2. GRAPHICS section, which contains information about the state of the view in RSCAD at time of storage.
3. DATA section with metadata about the model.
4. COMPONENT-ENUMERATION with information used by RSCAD for auto-enumeration.
5. SUBSYSTEM section with an enumeration of the subsystems in the model.

Components

Components are represented by a Type A 'COMPONENT-TYPE' node. The first line of the node contains the position and rotation of the component in multiple integer values, not in a key-value pair like all other information in the file. This is followed by a Type B 'PARAMETER' block with the values of the parameters of the component. The last block is a Type A 'ENUMERATION' block with the values of the enumeration parameters of the component.

Enumeration

Enumeration blocks contain four lines in the following format:

Line	Description	Format
1	Enumeration is active	true/false
2	Enumeration index	int
3	Enumeration type	Integer/Hex/ uppercase/lowercase
4	Enumeration string	string

Subsystems, Hierarchies and Groups

Every model contains one or multiple Subsystems with a canvas on which components are placed. In the Subsystem section, the subsystems are enumerated. The Subsystems themselves are SUBSYSTEM Type B nodes. In them, first the information about the canvas is defined and then the components are listed.

Subsystems can contain further canvases in Hierarchy components. Those are defined in the list of components like any other component, but are nested in a HIERARCHY Type B block. This block contains the Hierarchy component itself and the list of components that are nested in it.

Listing 3: Example of a Hierarchy node.

```
HIERARCHY-START:
COMPONENT_TYPE=HIERARCHY
    208 432 0 0 39
    PARAMETERS-START:
    Name    :box#
    x1      :-32
    y1      :-32
    x2      :32
    y2      :32
    PARAMETERS-END:
    ENUMERATION:
        true
        0
        Integer
        #
RUNTIME-OVERLAY-START: view VIEW-TYPE: DRAFT-VIEW VIEW-ID: test
RUNTIME-OVERLAY-END:
COMPONENT_TYPE=BUS
    240 144 0 0 7
    PARAMETERS-START:
    LW1     :3.0
    SCOL    :ORANGE
    DOCUMENT        :NO
```

(continues on next page)

(continued from previous page)

```
    x1      :-32
    y1      :-0
    x2      :32
    y2      :0
    PARAMETERS-END:
    ENUMERATION:
        true
        0
        Integer
        #
```

Groups

A group is a collection of components that can only be selected together in RSCAD. In .dfx files, groups contain the components in them in a a GROUP Type B node. The first component in the list is a GROUP component with only the 'COMPONENT-TYPE' line and the position line.

Listing 4: Example of a Group block.

```
GROUP-START:
COMPONENT_TYPE=GROUP
    1136 464 0 0 0
...
GROUP-END:
```

Components are added to groups by adding them to the corresponding group component with the add_component() method. Components in groups are only returned by the get_components() method if 'with_groups' is True or 'recursive' is set to True. The getConnectedTo() method and the connection graph contain the components in groups. However, the modify_component() and and remove_component() methods need to have 'recursive' set to True to modify the component in a group from the hierarchy/subsystem.

CHAPTER

TEN

COMPONENT BUILDER FILE FORMAT

10.1 Introduction

The Component Builder file format is a format used by RSCAD FX 1.0 and later to define a component type. The format consists of a simple text file structures by indentations. While the basic structure resembles *.dfx File Format* **Type A Nodes**, the format is far more complex than the simple key-value pairs used in .dfx files.

Like the CBuilder application, the file contains multiple sections defining different properties of the component type.

This documentation only documents the parts of the Component Builder format parsed and used by the current version of pyapi_rts. Other sections are omitted.

10.2 Structure of the Component Builder Format

The first line of the file is the string 'Component Builder' followed by the file version used. After that, different sections are defined by indentations, with an unindented line before describing the type of the block.

Listing 1: An example of a node in a Component Builder file

```
PARAMETERS:
    SECTION: "CONFIGURATION"
        LW1  "Bus thickness (Single Phase)"          "" 5 REAL 3.0 0.0
        SCOL "Bus Color"                    "RED;BLACK;BLUE;GREEN;CYAN;ORANGE;MAGENTA;
↪PINK;WHITE;BROWN;GOLD;VIOLET;YELLOW;LIGHT_GRAY" 10 TOGGLE   ORANGE
    DOCUMENT "include in print->parameters?"     "NO;YES" 10 TOGGLE 0
    SECTION: "HIDDEN PARAMETERS" false
      x1 "x1"  " " 4 INTEGER -32 0 0 false
      y1 "y1"  " " 4 INTEGER -0 0 0 false
      x2 "x2"  " " 4 INTEGER 32 0 0 false
      y2 "y2"  " " 4 INTEGER 0 0 0 false
```

10.2.1 Indentations

Indentations are used to define sections, but are inconsistent in a lot of cases. For this reason, some sections use custom parsers to build the block tree correctly. There can be no guarantee that sections and other hierarchies are recognized correctly, but there are warnings for unrecognized structures.

10.2.2 Parameter Section

The *PARAMETERS:* node defines the parameters of the component type. Parameters can be grouped into sections or be defined directly in the node. Grouped parameters are defined in a *SECTION: "<name>"* node.

The parameters themselves are defined in a line with the following format:

<key> "<description>" "<toggle>" <?> <type> <default> <min>? <max>? <enabled_condition>?

Notes:

- All parts marked with <...>? are optional.
- The *key* is the name of the parameter.
- The *description* is a short description of the parameter.
- The *toggle* lists the possible values of the parameter, separated by semicolons.
- The <*?*> can be ignored after parsing.
- The *type* is the type of the parameter.
- The *default* is the default value of the parameter.
- The *min* and *max* are the minimum and maximum values of the parameter.
- The *enabled_condition* is a logical expression that determines whether the parameter is enabled or not. The language used for conditions is described in the sections about conditions.

Example:

```
x1 "x1"  " " 4 INTEGER ^32 0 0 false
```

The following types are supported in the <type> field:

Type	Description
REAL	A real number
CHAR	A character
NAME	A string the enumerator is applied to
TOGGLE	A value from the <toggle> list
INTEGER	An integer
COLOR	A color supported by RSCAD
HEX	A hexadecimal number
FILE	A file path

10.2.3 Directives Section

The *DIRECTIVES:* node contains directives that are applied to the component type. They have the format **<KEY> = <VALUE>**.

The following directives are currently supported by pyapi_rts: STRETCHABLE

Value	Description
STRETCHABLE_DIAG_LINE	Can be stretched in any direction
STRETCHABLE_BOX	Horizontal/Vertical streching
STRETCHABLE_UP_DOWN_LINE	One strechable axis

10.2.4 Nodes Section

Nodes are defined in the *NODES:* section. Nodes are the points at which the component can connect to other components. In the Component Builder file, they are encoded in one line per node. Conditions are supported in this section as blocks, as described in the next section.

<name> <x-position> <y-position> <mode> [PHASE=<phase>]? <linked>? <...>?

Notes:

- Every <>? and []? entry is optional.
- The <*name*> is the name of the node.
- The <*x-position*> and <*y-position*> are relative to the component's origin.
- <*x-position*> and <*y-position*> can use parameter values with the '$key' syntax.
- The <*mode*> is the mode of the node and is ignored after parsing.
- The <*linked*> is the type of the node, pyapi_rts supports NAME_CONNECTED or a missing entry.
- The <*phase*> is the phase of the node, starting with 'PHASE='.
- The <...> is ignored after parsing.

Listing 2: Example

```
A_1  $x 0     EXTERNAL PHASE=A_PHASE NAME_CONNECTED:LINKED
```

10.2.5 Conditions

Conditions are boolean expressions using the value of parameters and logical operators. They are supported in multiple places in the Component Builder file and can be nested in other conditions, creating complex decision trees. This enables component to change their properties based on their parameters.

Conditions consist of the condition line and indented lines following it that are only active when the conditions evaluates to true.

Structure of the condition:

```
<#IF> <expression> <operator> <expression>
    content
<#ELSEIF> <expression> <operator> <expression>
    content
<#ELSE>
    content
#END
```

Notes:

- The *<#ELSEIF>* and *<#ELSE>* blocks are optional.
- The *<#END>* line is optional if another #IF condition follows.
- The content does not need to be indented if the block ends with a #END line.
- The *<expression>* is a parameter value or another logical expression.
- The *<operator>* is a logical operator.
- The *<content>* is active if the condition evaluates to true.

Supported operators on numbers:

Operator	Description
==	Equal with toggle evaluated as number
=	Equal on numbers
!=	Not Equal
<=	Smaller or equal
<=	Greater or equal
>	Greater
<	Smaller

The toggle operator '==' converts the value of the parameter to its index in the list of possible values for the parameter.

Supported operators on boolean expressions:

Operator	Description
&&	And
||	Or

CHAPTER

ELEVEN

COMPONENT EXTENSIONS

Warning: Difference to hooks **Component Extensions** extend the functionality of individual components. If you want to add new functionality to the whole API, you should use hooks if available.

11.1 Idea

Component Extensions enable the user to add new functionality on top of the existing, generic methods provided by the Component class and the ComponentBox class. This functionality is specified on a per-component basis, e.g. a new method specific to BUS components.

11.2 Extension Directory Structure

```
pyapi_rts/class_extractor/extensions
|
+---<extension_name>
    |
    +---<extension_for_BUS.py>
    |
    +---<extension_for_WIRE.py>
    |
    +---<shared_class.py>
    |
    +---<extension_name_test.py>
```

An extension directory can contain three types of files, subdirectories are ignored:

- **<extension_name>.py**: The extension class for one specified component type. This file must be a valid python class extending the Component class and contain a line with a *#EXTENDS: <component_type>* statement.
- **<shared_class>.py**: Shared code that can be used by multiple extensions. This file can contain any valid python code that does not contain an *EXTENDS* statement.
- **<extension_name_test.py>**: A test file that can be used to test the extension. Only one test class per extension is allowed.

11.3 Create a new Extension

Add a new extension to the pyapi_rts/class_extractor/extensions directory. The minimal extension must contain a **<component_extension>.py** file and a **<extension_name_test.py>** file. The **<component_extension>.py** file must contain a line with a

```
*#EXTENDS: <component_type>*
```

statement.

11.4 Imports in Extensions

Only the methods following the *#EXTENDS:* statement are copied to the component classes. The shared code is made available to the extension classes automatically, but it might still be useful to import them manually to get autocomoplete support during development.

If an import is required in the component extension class, the import statement has to be after the *#EXTENDS:* statement.

11.5 Testing the Extension

During the *ClassExtractor* run, the **<extension_name_test.py>** file is copied to the *tests/extensions* directory. The test can be executed with *poetry run pytest*, and is executed in the *extensions_test* stage of the GitLab pipeline, but not in the *test* stage.

11.6 Including and Excluding Extensions in the ClassExtractor

By default, all extensions are included in the ClassExtractor run. If the *-e / –extensions* option is set to 'false', extensions are ignored. If only certain extensions should be excluded, use the *–exclude-ext* option.

For more information, see the *ClassExtractor* documentation.

CHAPTER

TWELVE

EXTENSION HOOKS

Warning: **Extension Hooks** are distinct from the **Component Extensions**. Extension hooks are a way to extend the functionality of the API, for example the graph generation, and are called during the runtime. **Component Extensions** on the other hand extend new functionality to the components and are added to the classes during the **Class Extractor** rum and used by the user during runtime.

12.1 Introduction

Some component behavior and interactions between them are defined not at all or not in an easily readable way in the :ref: *Component Builder Format<component_builder_format>* files. The behavior needs to be implemented in the API manually. **Extension hooks** enable the addition of new functionality to the API in a structured way, for example adding new connections to the connection graph.

12.2 list of available Hooks

Name	Arguments	Returns	Function
Graph connections	components: list[Component] pos_dict link_dict	list[tuple[str, str]] *Graph connection between nodes with these UUIDs*	Adds new connections between components on the connection graph.
Link connections	components: list[Component]	list[tuple[str, str]] *New link_dict entries in form (name, UUID)*	Adds new entries to link_dict

12.3 Adding new Hooks

A hook is a Python class extending the *ComponentHook* class.

The hooks are class methods, so no state should be stored within the hook class.

Not all hook methods need to be implemented by a hook class.

Hooks need to be added to the `pyapi_rts/class_extractor/hooks` directory and are copied during the Class Extractor run.

12.3.1 Testing

As hooks are used as extension of the API functionality they can be tested with regular unit tests in the `tests` directory. The functionality implemented by hooks represents logic from RSCAD and is not optional, unlike Component Extensions. Nevertheless, it is advised to make clear that a test relies on a specific hook to make debugging easier.

12.4 Using Hooks vs. extending API

When should a hook be used as opposed to extending the core API?

A hook provides a simple entry point for extending specific functionality and groups them together in one file. This makes it particularly useful for more functionality that represents edge cases like connections that only apply to a few components in a specific arrangement. In contrast, extending the API itself is useful every time a change can be used for a larger set of components or for changes that are read directly from the *Component Builder* files.

As an example, the TLineHook class (*TLineHook*) is used to attach Tline components to the Tline Calculation Box. This connection is not specified in the Component Builder files and needs to be implemented manually, while only affecting a few specific arrangements of components.

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p