KicadModTree Documentation

Release 0.1

Thomas Pointhuber

Aug 12, 2020

Contents

KicadModTree is a framework which allows standalone creation KiCAD footprint.

CHAPTER 1

Overview

This framework is mainly based on the idea of scripted CAD systems (for example OpenSCAD). This means, everything is a node, and can be structured like a tree. In other words, you can group parts of the footprint, and translate them in any way you want. Also cloning & co. is no Problem anymore because of this concept.

To be able to create custom Nodes, I separated the system in two parts. Base nodes, which represents simple structures and also be used by KiCAD itself, and specialized nodes which alter the behaviour of base nodes (for example positioning), or represent a specialized usage of base nodes (for example RectLine).

When you serialize your footprint, the serialize command only has to handle base nodes, because all other nodes are based upon the base nodes. This allows us to write specialized nodes without worrying about the FileHandlers or other core systems. You simply create you special node, and the framework knows how to handle it seamlessly.

CHAPTER 2

Module Index

2.1 KicadModTree package

2.1.1 KicadModTree.nodes package

2.1.1.1 KicadModTree.nodes.base package

Those nodes represent the primitives which we can use to create footprints. They are 1:1 mappings to the corresponding types used in .kicad_mod files.

KicadModTree.nodes.base.Arc module

class KicadModTree.nodes.base.Arc.**Arc**(***kwargs*)

Bases: *KicadModTree.nodes.Node.Node*, KicadModTree.util.geometric_util.geometricArc

Add an Arc to the render tree

Parameters ****kwargs** – See below

Keyword Arguments

- *geometry* (geometricArc) alternative to using geometric parameters
- *center* (Vector2D) – center of arc
- *start* (Vector2D) – start point of arc
- *midpoint* (Vector2D) – alternative to start point point is on arc and defines point of equal distance to both arc ends arcs of this form are given as midpoint, center plus angle
- *end* (Vector2D) – alternative to angle arcs of this form are given as start, end and center
- *angle* (float) – angle of arc
- *layer* (str) – layer on which the arc is drawn (default: 'F.SilkS')

- *width* (`float`) – width of the arc line (default: None, which means auto detection)

Example

```
>>> from KicadModTree import *
>>> Arc(center=[0, 0], start=[-1, 0], angle=180, layer='F.SilkS')
```

cut (**other*)
cut line with given other element

Params

- ***other* (Line, Circle, Arc)** cut the element on any intersection with the given geometric element

KicadModTree.nodes.base.Circle module

class KicadModTree.nodes.base.Circle.**Circle**(***kwargs*)
Bases: *KicadModTree.nodes.Node.Node*, KicadModTree.util.geometric_util.geometricCircle

Add a Circle to the render tree

Parameters ****kwargs** – See below

Keyword Arguments

- *center* (`Vector2D`) – center of the circle
- *radius* (`float`) – radius of the circle
- *layer* (`str`) – layer on which the circle is drawn (default: 'F.SilkS')
- *width* (`float`) – width of the circle line (default: None, which means auto detection)

Example

```
>>> from KicadModTree import *
>>> Circle(center=[0, 0], radius=1.5, layer='F.SilkS')
```

cut (**other*)
cut line with given other element

Params

- ***other* (Line, Circle, Arc)** cut the element on any intersection with the given geometric element

rotate (*angle*, *origin=(0, 0)*, *use_degrees=True*)
Rotate circle around given origin

Params

- ***angle* (float)** rotation angle
- ***origin* (Vector2D)** origin point for the rotation. default: (0, 0)
- ***use_degrees* (boolean)** rotation angle is given in degrees. default:True

translate (*distance_vector*)
Translate circle

Params

- ***distance_vector*** **(Vector2D)** 2D vector defining by how much and in what direction to translate.

KicadModTree.nodes.base.Line module

class KicadModTree.nodes.base.Line.**Line**(***kwargs*)

Bases: *KicadModTree.nodes.Node.Node*, KicadModTree.util.geometric_util.geometricLine

Add a Line to the render tree

Parameters ****kwargs** – See below

Keyword Arguments

- *start* (Vector2D) – start point of the line
- *end* (Vector2D) – end point of the line
- *layer* (str) – layer on which the line is drawn (default: 'F.SilkS')
- *width* (float) – width of the line (default: None, which means auto detection)

Example

```
>>> from KicadModTree import *
>>> Line(start=[1, 0], end=[-1, 0], layer='F.SilkS')
```

cut(**other*)

cut line with given other element

Params

- ***other*** **(Line, Circle, Arc)** cut the element on any intersection with the given geometric element

KicadModTree.nodes.base.Model module

class KicadModTree.nodes.base.Model.**Model**(***kwargs*)

Bases: *KicadModTree.nodes.Node.Node*

Add a 3D-Model to the render tree

Parameters ****kwargs** – See below

Keyword Arguments

- *filename* (str) – name of the 3d-model file
- *at* (Vector3D) – position of the model (default: [0, 0, 0])
- *scale* (Vector3D) – scale of the model (default: [1, 1, 1])
- *rotate* (Vector3D) – rotation of the model (default: [0, 0, 0])

Example

```
>>> from KicadModTree import *
>>> Model(filename="example.3dshapes/example_footprint.wrl",
...       at=[0, 0, 0], scale=[1, 1, 1], rotate=[0, 0, 0])
```

KicadModTree.nodes.base.Pad module

class KicadModTree.nodes.base.Pad.**Pad**(***kwargs*)
Bases: *KicadModTree.nodes.Node.Node*

Add a Pad to the render tree

Parameters ****kwargs** – See below

Keyword Arguments

- *number* (int, str) – number/name of the pad (default: "")
- *type* (Pad.TYPE_THT, Pad.TYPE_SMT, Pad.TYPE_CONNECT, Pad.TYPE_NPTH) – type of the pad
- *shape* (Pad.SHAPE_CIRCLE, Pad.SHAPE_OVAL, Pad.SHAPE_RECT, SHAPE_ROUNDRECT,

Pad.SHAPE_TRAPEZE, SHAPE_CUSTOM) – shape of the pad

- *layers* (Pad.LAYERS_SMT, Pad.LAYERS_THT, Pad.LAYERS_NPTH) – layers on which are used for the pad
- *at* (Vector2D) – center position of the pad
- *rotation* (float) – rotation of the pad
- *size* (float, Vector2D) – size of the pad
- *offset* (Vector2D) – offset of the pad
- *drill* (float, Vector2D) – drill-size of the pad
- *radius_ratio* (float) – The radius ratio of the rounded rectangle. Ignored for every shape except round rect.
- *maximum_radius* (float) – The maximum radius for the rounded rectangle. If the radius produced by the radius_ratio parameter for the pad would exceed the maximum radius, the ratio is reduced to limit the radius. (This is useful for IPC-7351C compliance as it suggests 25% ratio with limit 0.25mm) Ignored for every shape except round rect.
- *round_radius_exact* (float) – Set an exact round radius for a pad. Ignored for every shape except round rect
- *round_radius_handler* (RoundRadiusHandler) – An instance of the RoundRadiusHandler class If this is given then all other round radius specifiers are ignored Ignored for every shape except round rect
- *solder_paste_margin_ratio* (float) – solder paste margin ratio of the pad (default: 0)
- *solder_paste_margin* (float) – solder paste margin of the pad (default: 0)
- *solder_mask_margin* (float) – solder mask margin of the pad (default: 0)
- *x_mirror* ([int, float](mirror offset)) – mirror x direction around offset "point"
- *y_mirror* ([int, float](mirror offset)) – mirror y direction around offset "point"

Example

```
>>> from KicadModTree import *
>>> Pad(number=1, type=Pad.TYPE_THT, shape=Pad.SHAPE_RECT,
...     at=[0, 0], size=[2, 2], drill=1.2, layers=Pad.LAYERS_THT)
```

addPrimitive(*p*)
add a primitve to a custom pad

Parameters **p** – the primitive to add

rotate(*angle*, *origin=(0, 0)*, *use_degrees=True*)
Rotate pad around given origin

Params

- ***angle*** **(float)** rotation angle
- ***origin*** **(Vector2D)** origin point for the rotation. default: (0, 0)
- ***use_degrees*** **(boolean)** rotation angle is given in degrees. default:True

translate(*distance_vector*)
Translate pad

Params

- ***distance_vector*** **(Vector2D)** 2D vector defining by how much and in what direction to translate.

class KicadModTree.nodes.base.Pad.**RoundRadiusHandler**(***kwargs*)
Bases: object

Handles round radius setting of a pad

Parameters ****kwargs** – See below

Keyword Arguments

- *radius_ratio* (float [0 <= r <= 0.5]) – The radius ratio of the rounded rectangle. (default set by default_radius_ratio)
- *maximum_radius* (float) – The maximum radius for the rounded rectangle. If the radius produced by the radius_ratio parameter for the pad would exceed the maximum radius, the ratio is reduced to limit the radius. (This is useful for IPC-7351C compliance as it suggests 25% ratio with limit 0.25mm)
- *round_radius_exact* (float) – Set an exact round radius for a pad.
- *default_radius_ratio* (float [0 <= r <= 0.5]) – This parameter allows to set the default radius ratio (backwards compatibility option for chamfered pads)

getRadiusRatio(*shortest_sidelength*)
get the resulting round radius ratio

Parameters **shortest_sidelength** – shortest sidelength of a pad

Returns the resulting round radius ratio to be used for the pad

getRoundRadius(*shortest_sidelength*)
get the resulting round radius

Parameters **shortest_sidelength** – shortest sidelength of a pad

Returns the resulting round radius to be used for the pad

limitMaxRadius(*limit*)
Set a new maximum limit

Parameters **limit** – the new limit.

roundingRequested()
Check if the pad has a rounded corner

Returns True if rounded corners are required

KicadModTree.nodes.base.Polygon module

class KicadModTree.nodes.base.Polygon.**Polygon**(***kwargs*)
Bases: *KicadModTree.nodes.Node.Node*

Add a Polygon to the render tree

Parameters ****kwargs** – See below

Keyword Arguments

- *polygon* (list(Point)) – outer nodes of the polygon
- *layer* (str) – layer on which the line is drawn (default: 'F.SilkS')
- *width* (float) – width of the line (default: None, which means auto detection)
- *x_mirror* ([int, float](mirror offset)) – mirror x direction around offset "point"
- *y_mirror* ([int, float](mirror offset)) – mirror y direction around offset "point"

Example

```
>>> from KicadModTree import *
>>> Polygon(nodes=[[-2, 0], [0, -2], [4, 0], [0, 2]], layer='F.SilkS')
```

cut(*other*)
Cut other polygon from this polygon

More details see PolygonPoints.cut docstring.

Parameters **other** – the other polygon

rotate(*angle*, *origin=(0, 0)*, *use_degrees=True*)
Rotate polygon around given origin

Params

- ***angle*** **(float)** rotation angle
- ***origin*** **(Vector2D)** origin point for the rotation. default: (0, 0)
- ***use_degrees*** **(boolean)** rotation angle is given in degrees. default:True

translate(*distance_vector*)
Translate polygon

Params

- ***distance_vector*** **(Vector2D)** 2D vector defining by how much and in what direction to translate.

KicadModTree.nodes.base.Text module

class KicadModTree.nodes.base.Text.**Text**(***kwargs*)
Bases: *KicadModTree.nodes.Node.Node*

Add a Line to the render tree

Parameters ****kwargs** – See below

Keyword Arguments

- *type* (str) – type of text
- *text* (str) – text which is been visualized
- *at* (Vector2D) – position of text
- *rotation* (float) – rotation of text (default: 0)
- *mirror* (bool) – mirror text (default: False)
- *layer* (str) – layer on which the text is drawn (default: 'F.SilkS')
- *size* (Vector2D) – size of the text (default: [1, 1])
- *thickness* (float) – thickness of the text (default: 0.15)
- *hide* (bool) – hide text (default: False)

Example

```
>>> from KicadModTree import *
>>> Text(type='reference', text='REF**', at=[0, -3], layer='F.SilkS')
>>> Text(type='value', text="footprint name", at=[0, 3], layer='F.Fab')
>>> Text(type='user', text='test', at=[0, 0], layer='Cmts.User')
```

rotate(*angle*, *origin=(0, 0)*, *use_degrees=True*)
Rotate text around given origin

Params

- ***angle*** **(float)** rotation angle
- ***origin*** **(Vector2D)** origin point for the rotation. default: (0, 0)
- ***use_degrees*** **(boolean)** rotation angle is given in degrees. default:True

translate(*distance_vector*)
Translate text

Params

- ***distance_vector*** **(Vector2D)** 2D vector defining by how much and in what direction to translate.

2.1.1.2 KicadModTree.nodes.specialized package

To simpilfy the creation on footprints, we have special classes which are build onto the base nodes. special nodes are converted to base nodes when creating the footprint, and allows us to create much more complex shapes with as little boilerplate code as possible.

KicadModTree.nodes.specialized.PolygoneLine module

class KicadModTree.nodes.specialized.PolygoneLine.**PolygoneLine**(***kwargs*)
Bases: *KicadModTree.nodes.Node.Node*

Add a Polygone Line to the render tree

Parameters ****kwargs** – See below

Keyword Arguments

- *polygone* (list(Point)) – edges of the polygone
- *layer* (str) – layer on which the polygone is drawn (default: 'F.SilkS')
- *width* (float) – width of the line (default: None, which means auto detection)

Example

```
>>> from KicadModTree import *
>>> PolygoneLine(polygone=[[0, 0], [0, 1], [1, 1], [0, 0]], layer='F.SilkS')
```

getVirtualChilds()
Get virtual childs of this node

KicadModTree.nodes.specialized.RectLine module

class KicadModTree.nodes.specialized.RectLine.**RectLine**(***kwargs*)
Bases: *KicadModTree.nodes.specialized.PolygoneLine.PolygoneLine*

Add a Rect to the render tree

Parameters ****kwargs** – See below

Keyword Arguments

- *start* (Vector2D) – start edge of the rect
- *end* (Vector2D) – end edge of the rect
- *layer* (str) – layer on which the rect is drawn
- *width* (float) – width of the outer line (default: None, which means auto detection)
- *offset* (Vector2D, float) – offset of the rect line to the specified one

Example

```
>>> from KicadModTree import *
>>> RectLine(start=[-3, -2], end=[3, 2], layer='F.SilkS')
```

KicadModTree.nodes.specialized.RectFill module

class KicadModTree.nodes.specialized.RectFill.**RectFill**(***kwargs*)
Bases: *KicadModTree.nodes.Node.Node*

Add the filling of a Rect to the render tree

Normally, this class isn't needed, because FilledRect combines RectLine with RectFill

Parameters ****kwargs** – See below

Keyword Arguments

- *start* (`Vector2D`) – start edge of the rect fill
- *end* (`Vector2D`) – end edge of the rect fill
- *layer* (`str`) – layer on which the rect fill is drawn (default: 'F.SilkS')
- *width* (`float`) – width of the filling lines (default: 0.12)

Example

```
>>> from KicadModTree import *
>>> RectFill(start=[-3, -2], end=[3, 2], layer='F.SilkS')
```

getVirtualChilds()
Get virtual childs of this node

KicadModTree.nodes.specialized.FilledRect module

class KicadModTree.nodes.specialized.FilledRect.**FilledRect**(***kwargs*)
Bases: *KicadModTree.nodes.Node.Node*

Add a Filled Rect to the render tree

Combines `RectLine` and `RectFill` into one class for simpler handling

Parameters ****kwargs** – See below

Keyword Arguments

- *start* (`Vector2D`) – start edge of the rect
- *end* (`Vector2D`) – end edge of the rect
- *layer* (`str`) – layer on which the rect is drawn (default: 'F.SilkS')
- *width* (`float`) – width of the outer line (default: 0.15)

Example

```
>>> from KicadModTree import *
>>> FilledRect(start=[-3, -2], end=[3, 2], layer='F.SilkS')
```

getVirtualChilds()
Get virtual childs of this node

KicadModTree.nodes.specialized.PadArray module

class KicadModTree.nodes.specialized.PadArray.**PadArray**(***kwargs*)
Bases: *KicadModTree.nodes.Node.Node*

Add a row of Pads

Simplifies the handling of pads which are rendered in a specific form

Parameters ****kwargs** – See below

Keyword Arguments

- *start* (`Vector2D`) – start edge of the pad array
- *center* (`Vector2D`) – center pad array around specific point

- *pincount* (`int`) – number of pads to render
- *spacing* (`Vector2D`, `float`) – offset between rendered pads
- *x_spacing* (`float`) – x offset between rendered pads
- *y_spacing* (`float`) – y offset between rendered pads
- *initial* (`int`) – name of the first pad
- *increment* (`int, function(previous_number)`) – declare how the name of the follow up is calculated
- *type* (`Pad.TYPE_THT`, `Pad.TYPE_SMT`, `Pad.TYPE_CONNECT`, `Pad.TYPE_NPTH`) – type of the pad
- *shape* (`Pad.SHAPE_CIRCLE`, `Pad.SHAPE_OVAL`, `Pad.SHAPE_RECT`, `Pad.SHAPE_TRAPEZE`, ...) – shape of the pad
- *rotation* (`float`) – rotation of the pad
- *size* (`float`, `Vector2D`) – size of the pad
- *offset* (`Vector2D`) – offset of the pad
- *drill* (`float`, `Vector2D`) – drill-size of the pad
- *solder_paste_margin_ratio* (`float`) – solder paste margin ratio of the pad
- *layers* (`Pad.LAYERS_SMT`, `Pad.LAYERS_THT`, `Pad.LAYERS_NPTH`) – layers on which are used for the pad
- *chamfer_corner_selection_first* (`[bool, bool, bool, bool]`) Select which corner should be chamfered for the first pad. (default: None)
- *chamfer_corner_selection_last* (`[bool, bool, bool, bool]`) Select which corner should be chamfered for the last pad. (default: None)
- *chamfer_size* (`float`, `Vector2D`) – size for the chamfer used for the end pads. (default: None)
- *end_pads_size_reduction* (`dict with keys x-,x+,y-,y+`) – size is reduced on the given side. (size reduced plus center moved.)
- *tht_pad1_shape* (`Pad.SHAPE_RECT`, `Pad.SHAPE_ROUNDRECT`, ...) – shape for marking pad 1 for through hole components. (deafult: `Pad.SHAPE_ROUNDRECT`)
- *tht_pad1_id* (`int, string`) – pad number used for "pin 1" (default: 1)
- *exclude_pin_list* (`int, Vector1D`) – which pin number should be skipped"

Example

```
>>> from KicadModTree import *
>>> PadArray(pincount=10, spacing=[1,-1], center=[0,0], initial=5, increment=2,
...          type=Pad.TYPE_SMT, shape=Pad.SHAPE_RECT, size=[1,2], layers=Pad.
↪LAYERS_SMT)
```

getVirtualChilds()
Get virtual childs of this node

KicadModTree.nodes.specialized.Rotation module

class KicadModTree.nodes.specialized.Rotation.**Rotation**(*r*)
Bases: *KicadModTree.nodes.Node.Node*

Apply rotation to the child tree

Parameters **r** (float) – angle which the child should rotate

Example

```
>>> from KicadModTree import *
>>> Rotation(90)
```

getRealPosition(*coordinate*, *rotation=None*)
return position of point after applying all transformation and rotation operations

KicadModTree.nodes.specialized.Translation module

class KicadModTree.nodes.specialized.Translation.**Translation**(*x*, *y*)
Bases: *KicadModTree.nodes.Node.Node*

Apply translation to the child tree

Parameters

- **x** (float) – change of x coordinate
- **y** (float) – change of y coordinate

Example

```
>>> from KicadModTree import *
>>> Translation(1, 2)
```

getRealPosition(*coordinate*, *rotation=None*)
return position of point after applying all transformation and rotation operations

2.1.1.3 KicadModTree.nodes.Footprint module

class KicadModTree.nodes.Footprint.**Footprint**(*name*)
Bases: *KicadModTree.nodes.Node.Node*

Root Node to generate KicadMod

setAttribute(*value*)

setDescription(*description*)

setMaskMargin(*value*)

setName(*name*)

setPasteMargin(*value*)

setPasteMarginRatio(*value*)

setTags(*tags*)

2.1.1.4 KicadModTree.nodes.Node module

exception KicadModTree.nodes.Node.**MultipleParentsError**(*message*)
Bases: exceptions.RuntimeError

class KicadModTree.nodes.Node.**Node**
Bases: object

append(*node*)
add node to child

calculateBoundingBox(*outline=None*)

copy()

extend(*nodes*)
add list of nodes to child

getAllChilds()
Get virtual and normal childs of this node

getCompleteRenderTree(*rendered_nodes=None*)
print virtual render tree

getNormalChilds()
Get all normal childs of this node

getParent()
get Parent Node of this Node

getRealPosition(*coordinate*, *rotation=None*)
return position of point after applying all transformation and rotation operations

getRenderTree(*rendered_nodes=None*)
print render tree

getRootNode()
get Root Node of this Node

getVirtualChilds()
Get virtual childs of this node

insert(*node*)
moving all childs into the node, and using the node as new parent of those childs

remove(*node*)
remove child from node

serialize()

exception KicadModTree.nodes.Node.**RecursionDetectedError**(*message*)
Bases: exceptions.RuntimeError

2.1.2 KicadModTree.util package

2.1.2.1 KicadModTree.util.kicad_util module

class KicadModTree.util.kicad_util.**SexprSerializer**(*sexpr*)
Bases: object

Converts a nested python list into a sexpr syntax which can be parsed by KiCad

NEW_LINE
alias of `__builtin__.object`

primitive_to_string(*primitive*)

sexpr_to_string(*sexpr*, *prefix=None*)

`KicadModTree.util.kicad_util.`**formatFloat**(*val*)
return well formated float

`KicadModTree.util.kicad_util.`**formatTimestamp**(*timestamp=None*)

`KicadModTree.util.kicad_util.`**lispString**(*string*)
add quotation marks to string, when it include a white space or is empty

`KicadModTree.util.kicad_util.`**lispTokenizer**(*input*)
Convert a string of characters into a list of tokens.

`KicadModTree.util.kicad_util.`**parseLispString**(*input*)

`KicadModTree.util.kicad_util.`**parseTimestamp**(*timestamp*)

2.1.3 KicadModTree.FileHandler module

class `KicadModTree.FileHandler.`**FileHandler**(*kicad_mod*)
Bases: `object`

some basic methods to write footprints, and which is the base class of footprint writer implementations

Parameters **kicad_mod** (`KicadModTree.Footprint`) – Main object representing the footprint

Example

```
>>> from KicadModTree import *
>>> kicad_mod = Footprint("example_footprint")
>>> file_handler = KicadFileHandler(kicad_mod)  # KicadFileHandler is a␣
↪implementation of FileHandler
>>> file_handler.writeFile('example_footprint.kicad_mod')
```

serialize(***kwargs*)
Get a valid string representation of the footprint in the specified format

Example

```
>>> from KicadModTree import *
>>> kicad_mod = Footprint("example_footprint")
>>> file_handler = KicadFileHandler(kicad_mod)  # KicadFileHandler is a␣
↪implementation of FileHandler
>>> print(file_handler.serialize())
```

writeFile(*filename*, ***kwargs*)
Write the output of FileHandler.serialize to a file

Parameters **filename** (`str`) – path of the output file

Example

```
>>> from KicadModTree import *
>>> kicad_mod = Footprint("example_footprint")
```

(continues on next page)

(continued from previous page)

```
>>> file_handler = KicadFileHandler(kicad_mod)  # KicadFileHandler is a␣
↪implementation of FileHandler
>>> file_handler.writeFile('example_footprint.kicad_mod')
```

2.1.4 KicadModTree.KicadFileHandler module

class KicadModTree.KicadFileHandler.**KicadFileHandler**(*kicad_mod*)

Bases: *KicadModTree.FileHandler.FileHandler*

Implementation of the FileHandler for .kicad_mod files

Parameters **kicad_mod** (KicadModTree.Footprint) – Main object representing the footprint

Example

```
>>> from KicadModTree import *
>>> kicad_mod = Footprint("example_footprint")
>>> file_handler = KicadFileHandler(kicad_mod)
>>> file_handler.writeFile('example_footprint.kicad_mod')
```

serialize(***kwargs*)

Get a valid string representation of the footprint in the .kicad_mod format

Example

```
>>> from KicadModTree import *
>>> kicad_mod = Footprint("example_footprint")
>>> file_handler = KicadFileHandler(kicad_mod)
>>> print(file_handler.serialize())
```

writeFile(*filename*, ***kwargs*)

Write the output of FileHandler.serialize to a file

Parameters **filename** (str) – path of the output file

Example

```
>>> from KicadModTree import *
>>> kicad_mod = Footprint("example_footprint")
>>> file_handler = KicadFileHandler(kicad_mod)  # KicadFileHandler is a␣
↪implementation of FileHandler
>>> file_handler.writeFile('example_footprint.kicad_mod')
```

2.1.5 KicadModTree.ModArgparser module

class KicadModTree.ModArgparser.**ModArgparser**(*footprint_function*)

Bases: object

A general data loading class, which allows us to specify parts using .yml or .csv files.

Using this class allows us to seperate between the implementation of a footprint generator, and the data which represents a single footprint. To do so, we need to define which parameters are expected in those data-files.

To improve the usablity of this class, it is able to do type checks of provided parameters, as well as defining default values and do a simple check if a parameter can be considered as required or optional.

Parameters **footprint_function** (function reference) – A function which is called for every footprint we want to generate

Example

```
>>> from KicadModTree import *
>>> def footprint_gen(args):
...     print("create footprint: {}".format(args['name']))
...
>>> parser = ModArgparser(footprint_gen)
>>> parser.add_parameter("name", type=str, required=True)  # the root node of .
↪yml files is parsed as name
>>> parser.add_parameter("datasheet", type=str, required=False)
>>> parser.add_parameter("courtyard", type=float, required=False, default=0.25)
>>> parser.add_parameter("pincount", type=int, required=True)
>>> parser.run()  # now run our script which handles the whole part of parsing␣
↪the files
```

add_parameter(*name*, ***kwargs*)

Add a parameter to the ModArgparser

Parameters

- **name** (str) – name of the argument
- ****kwargs** – See below

Keyword Arguments

- *type* (type) – type of the argument
- *required* (bool) – is the argument required or optional
- *default* – a default value which is used when there is no value defined

Example

```
>>> from KicadModTree import *
>>> def footprint_gen(args):
...     print("create footprint: {}".format(args['name']))
...
>>> parser = ModArgparser(footprint_gen)
>>> parser.add_parameter("name", type=str, required=True)  # the root node of␣
↪.yml files is parsed as name
>>> parser.add_parameter("datasheet", type=str, required=False)
>>> parser.add_parameter("courtyard", type=float, required=False, default=0.
↪25)
```

run()

Execute the ModArgparser and run all tasks defined via the commandline arguments of this script

This method parses the commandline arguments to determine which actions to take. Beside of parsing .yml and .csv files, it also allows us to output example files.

```
>>> from KicadModTree import *
>>> def footprint_gen(args):
...     print("create footprint: {}".format(args['name']))
...
>>> parser = ModArgparser(footprint_gen)
```

(continues on next page)

(continued from previous page)

```
>>> parser.add_parameter("name", type=str, required=True)  # the root node of␣
↪.yml files is parsed as name
>>> parser.run()  # now run our script which handles the whole part of␣
↪parsing the files
```

exception KicadModTree.ModArgparser.**ParserException**
Bases: exceptions.Exception

2.1.6 KicadModTree.Vector module

class KicadModTree.Vector.**Vector2D**(*coordinates=None*, *y=None*)
Bases: object

Representation of a 2D Vector in space

Example

```
>>> from KicadModTree import *
>>> Vector2D(0, 0)
>>> Vector2D([0, 0])
>>> Vector2D((0, 0))
>>> Vector2D({'x': 0, 'y':0})
>>> Vector2D(Vector2D(0, 0))
```

__add__(*value*)

__copy__()

__dict__ = dict_proxy({'_Vector2D__arithmetic_parse': <staticmethod object>, '__modul

__div__(*value*)

__eq__(*other*)
x.__eq__(y) <==> x==y

__getitem__(*key*)

__iadd__(*value*)

__init__(*coordinates=None*, *y=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

__isub__(*value*)

__iter__()

__len__()

__module__ = 'KicadModTree.Vector'

__mul__(*value*)

__ne__(*other*)
x.__ne__(y) <==> x!=y

__neg__()

__repr__() <==> *repr(x)*

__setitem__(*key*, *item*)

__str__() <==> *str(x)*

__sub__(*value*)

__truediv__(*obj*)

__weakref__
list of weak references to the object (if defined)

distance_to(*value*)
Distance between this and another point

Parameters **value** – the other point

Returns distance between self and other point

static from_homogeneous(*source*)
Recover 2d vector from its homogeneous representation

Params

- ***source*** **(Vector3D)** 3d homogeneous representation

static from_polar(*radius*, *angle*, *origin=(0, 0)*, *use_degrees=True*)
Generate a vector from its polar representation

Params

- ***radius*** **(float)** lenght of the vector
- ***angle*** **(float)** angle of the vector
- ***origin*** **(Vector2D)** origin point for polar conversion. default: (0, 0)
- ***use_degrees*** **(boolean)** angle in degrees. default:True

render(*formatcode*)

rotate(*angle*, *origin=(0, 0)*, *use_degrees=True*)
Rotate vector around given origin

Params

- ***angle*** **(float)** rotation angle
- ***origin*** **(Vector2D)** origin point for the rotation. default: (0, 0)
- ***use_degrees*** **(boolean)** rotation angle is given in degrees. default:True

round_to(*base*)
Round to a specific base (like it's required for a grid)

Parameters **base** – base we want to round to

Returns rounded point

```
>>> from KicadModTree import *
>>> Vector2D(0.1234, 0.5678).round_to(0.01)
```

to_dict()

to_homogeneous()
Get homogeneous representation

to_polar(*origin=(0, 0)*, *use_degrees=True*)
Get polar representation of the vector

Params

- ***origin*** **(Vector2D)** origin point for polar conversion. default: (0, 0)

- ***use_degrees*** **(boolean)** angle in degrees. default:True

class KicadModTree.Vector.**Vector3D**(*coordinates=None*, *y=None*, *z=None*)

Bases: *KicadModTree.Vector.Vector2D*

Representation of a 3D Vector in space

Example

```
>>> from KicadModTree import *
>>> Vector3D(0, 0, 0)
>>> Vector3D([0, 0, 0])
>>> Vector3D((0, 0, 0))
>>> Vector3D({'x': 0, 'y':0, 'z':0})
>>> Vector3D(Vector2D(0, 0))
>>> Vector3D(Vector3D(0, 0, 0))
```

__add__(*value*)

__copy__()

__div__(*value*)

__eq__(*other*)
x.__eq__(y) <==> x==y

__getitem__(*key*)

__iadd__(*value*)

__init__(*coordinates=None*, *y=None*, *z=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

__isub__(*value*)

__iter__()

__len__()

__module__ = 'KicadModTree.Vector'

__mul__(*value*)

__ne__(*other*)
x.__ne__(y) <==> x!=y

__neg__()

__repr__() <==> *repr(x)*

__setitem__(*key*, *item*)

__str__() <==> *str(x)*

__sub__(*value*)

__truediv__(*obj*)

cross_product(*other*)

dot_product(*other*)

render(*formatcode*)

round_to(*base*)
Round to a specific base (like it's required for a grid)

Parameters **base** – base we want to round to

Returns rounded point

```
>>> from KicadModTree import *
>>> Vector3D(0.123, 0.456, 0.789).round_to(0.01)
```

to_dict()

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

Symbols

A

C

D

E

F

G

I

K

L

M

N

P

R

S

T

V

W