PyCell4Klayout

Release 0.1

IHP Authors

Jun 21, 2024

CONTENTS:

PyCell4Klayout is a Python library for supporting the PyCell API under the layout tool Klayout.

Note: This project is under active development.

CHAPTER

ONE

REFERENCED DOCUMENTS

No.	Doc ID-Number	Title
[1]	2021.09	Synopsys ‘Python API Reference Manual’

CHAPTER

TWO

INTRODUCTION

This reference manual documents the PyCell4Klayout API (Application Programming Interface) which is used to create parameterized cells within the Klayout design environment. This Klayout design environment makes use of the popular open-source Python programming language to provide a highly productive design environment for creating parameterized cells for analog layout design purposes. This PyCell4Klayout API provides a large number of classes and methods which are specialized for layout designs. By using these Python classes to provide powerful, high-level layout design abstractions, the Klayout design environment is extremely productive.

2.1 Basic Structure

The basic PyCell4Klayout system is built upon a set of base classes, from which the basic design and layout objects are generated through the PyCell Python API. These base classes are made accessible through the PyCell Python API, but do not have their own creation methods. Instead, other objects which are derived from these base classes can be constructed through the use of the PyCell Python API.

CHAPTER

THREE

IMPLEMENTATION DETAILS

3.1 Klayout

Klayout is an open-source EDA layout tool with a rich set of functionalities like layout editing, DRC, LVS, PCells. scripting and so on. The feature set and the Klayout GUI are implemented in an object oriented C++ core using the Qt library. Klayout supports the progamming languages Ruby and Python for scripting. The overall principle is that most of the C++ core classes have pendants with the same name in both the Ruby and Python scripting world. In the scripting world these pendants are just proxies which delegates an API-call to the C++ pendant which implements the API-call (language binding). This mechanism also covers the needed object lifetime management as well as the parameter type conversion back- and forward in the both directions between C++- and scripting-world (Marshalling).

3.1.1 PCell support

Klayout uses an own schema for supporting parametrized cells. The creation of PCells are covered in principle by the C++ core class PCellDeclaration. This class is also available in the Klayout Python namespace. The class PCellDeclaration as a PCell base class defines an API with some virtual function which must be implemented by a subclass to build a new PCell. This implemented functions are then called by the Klayout runtime engine to create/manage a new PCell. The most important of these functions are:

- *get_parameters*: Returns a list of parameter declarations for the PCell
- *coerce_parameters*: Modifies the parameters to match the requirements of the PCell
- *produce*: The production callback which creates the PCell layout

One method to create a PCell is subclassing a new class from PCellDeclaration and implement this set of specific function. This can be done in a Ruby- as well as Python-script.

3.2 Synopsys PyCell API

The PyCell API is build by a hierarchy of Python classes with a defined API [1]. The PyCell API classes can be roughly divided into the following groups:

- Basic geometry classes: Point, Box, Segement, Font, ...
- Physical component classes: Shape, Arc, Line, Dot,...
- Physical component related classes: Contact, AbutContact, Bar, DeviceContact, Via, ...
- Physical component reference classes: GroupingRef, InstanceRef, PolygonRef, ...
- Connectivity classes: SignalType, TermType, Net, Term, Pin, Layer, Tech, ...
- Parameter classes: ParamArray, ChoiceConstraint, RangeConstraint, ...

- PCell creation classes: Dlo, DloGen, Lib

3.2.1 PyCell creation principle

The DloGen class is the base class for all types of PCell generators. Any PCell generator would be derived from this base class. A PCell generator class which is derived from the DloGen base class must implement the following methods:

- *defineParamSpecs*: defines the parameters, including default values and constraints, for this PyCell
- *setupParams*: extracts the value for the parameters specified by the user for this PyCell
- *genLayout*: generates the actual physical layout for this PyCell

3.3 Wrapper solution approach

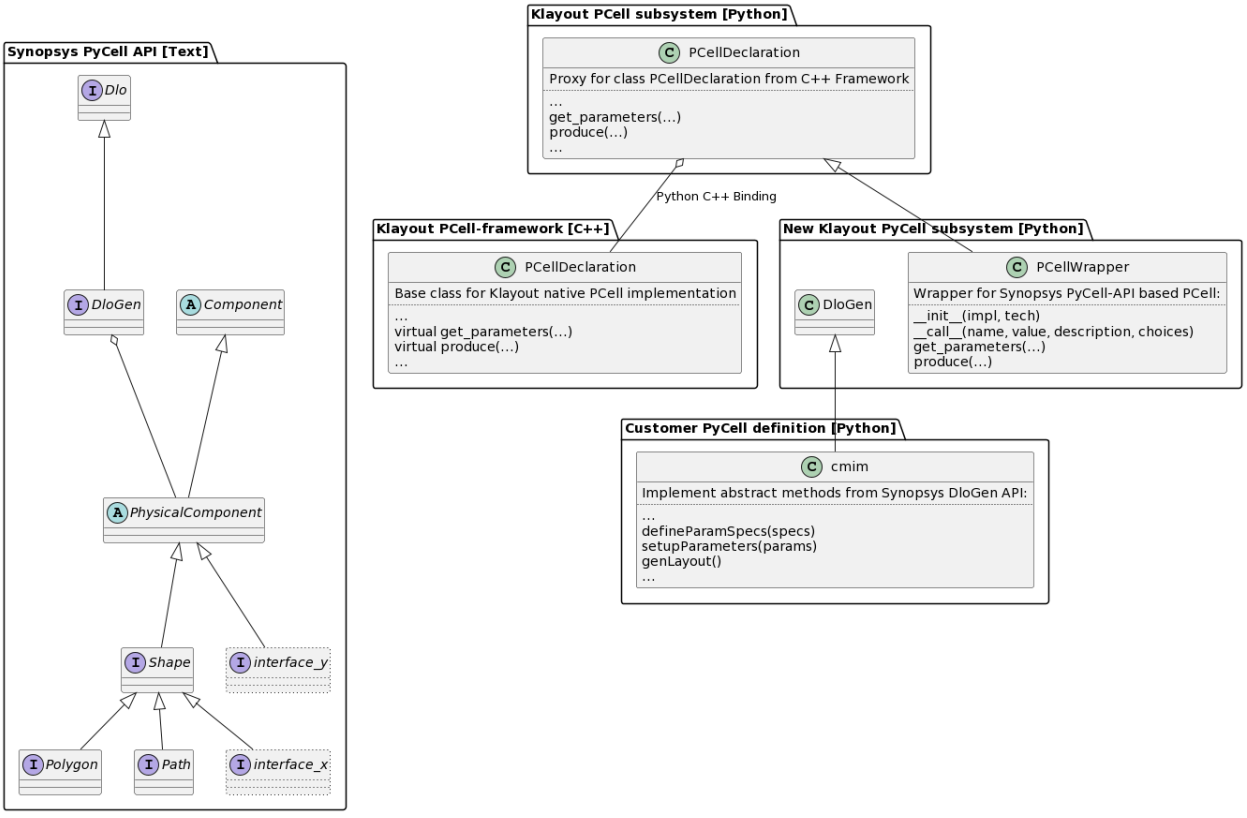

Fig. 1: PyCell wrapper structural architecture overview

3.4 PDK technology information handling

All parameters of a specific technology are encoded as JSON and can be found in a PDK specific file *technology*_tech.json, e.g. sg13g2_tech.json. The layer properties name, value and purpose are aquiered from a XML file *technology*.lyp, e.g. sg13g2.lyp. The lyp-files are so called *layer property files* and will be used by Klayout for naming layer, coloring etc., for details see the Klayout documentation. The lyp-files are expected under the path *path_to_technology_tech.json/../../tech/*. A different lyp-file can be set with the environment variable *KLAYOUT_LYP_FILE*.

CHAPTER

FOUR

API REFERENCE

This page contains auto-generated API reference documentation[1].

4.1 font

4.1.1 Module Contents

Classes

Font

class font.**Font**

Bases: object

EURO_STYLE = 1

FIXED = 2

GOTHIC = 3

MATH = 4

MIL_SPEC = 5

ROMAN = 6

SCRIPT = 7

STICK = 8

SWEDISH = 9

classmethod getMembers()

calcBBox(*text*, *origin*, *height*, *location=Location.UPPER_LEFT*, *orient=Orientation.R0*, *overbar=False*)

[1] Created with sphinx-autoapi

4.2 namemapper

4.2.1 Module Contents

Classes

NameMapper

class namemapper.**NameMapper**(*obj: object = None*)

Bases: object

4.3 box

4.3.1 Module Contents

Classes

Box

class box.**Box**(*l=INT_MAX*, *b=INT_MAX*, *r=INT_MIN*, *t=INT_MIN*)

Bases: object

property bottom

property left

property right

property top

abut(*dir*, *refBox*, *align=True*)

alignEdge(*dir*, *refBox*, *refDir=None*, *offset=None*)

alignEdgeToCoord(*dir*, *coord*)

alignEdgeToPoint(*dir*, *point*)

alignLocation(*loc*, *refBox*, *refLoc=None*, *offset=None*)

alignLocationToPoint(*loc*, *pt*)

centerCenter()

centerLeft()

centerRight()

clone(*nameMap: cni.namemapper.NameMapper = NameMapper()*, *netMap: cni.namemapper.NameMapper = NameMapper()*)

contains(*box*, *incEdges=True*)

containsPoint(*p*, *incEdges=True*)

destroy()

expand(*coord*)

expandDir(*dir*, *coord*)

expandForMinArea(*dir*, *minArea*, *grid=None*)

expandForMinWidth(*dir*, *minWidth*, *grid=None*)

expandToGrid(*grid*, *dir=None*)

fix()

getArea()

getCenter()

getCenterX()

getCenterY()

getCoord(*dir*)

getDimension(*dir*)

getHeight()

getLeft()

getLocationPoint(*loc*)

getLocationPoint(*dir*)

getPoints()

getRange(*dir*)

getRangeX()

getRangeY()

getRight()

getSpacing(*dir*, *refBox*)

getTop()

getWidth()

hasNoArea()

init()

intersect(*box*)

intersect(*box*, *dir*)

isInverted()

isNormal()

limit(*point*)

lowerCenter()

lowerLeft()

lowerRight()

merge(*box*, *dir*)

mergePoint(*p*)

mirrorX(*yCoord=0*)

mirrorY(*xCoord=0*)

moveBy(*dx: float*, *dy: float*) → None

moveTo(*destination*, *loc=Location.CENTER_CENTER*)

moveTowards(*dir*, *d*)

overlaps(*box*, *incEdges=True*)

place(*dir*, *refBox*, *distance*, *align=True*)

removeRegion(*box*)

rotate90(*origin=None*)

rotate180(*origin=None*)

rotate270(*origin=None*)

set(*b*)

set(*b*, *dir=None*)

set(*lowerLeft*, *upperRight*)

set(*left*, *bottom*, *right*, *top*)

setBottom(*v*)

setCenter(*point*)

setCenterY(*v*)

setCoord(*dir*, *coord*)

setDimension(*coord*, *dir*)

setBottom(*v*)

setHeight(*height*)

setLocationPoint(*loc*, *pt*)

setRange(*dir*, *range*)

setRangeX(*range*)

setRangeY(*range*)

setRect(*rect*)

setRight(*v*)

setTop(*v*)

setWidth(*width*)

snap(*grid*, *snapType=None*)

snapX(*grid*, *snapType=None*)

snapY(*grid*, *snapType=None*)

snapTowards(*grid*, *dir*)

transform(*transform: cni.transform.Transform*) → None

upperCenter()

upperLeft()

upperRight()

4.4 grouping

4.4.1 Module Contents

Classes

Grouping	Helper class that provides a standard way to create an ABC using

class grouping.**Grouping**(*name: str = ''*, *components: cni.physicalComponent.PhysicalComponent = None*)

Bases: cni.physicalComponent.PhysicalComponent

Helper class that provides a standard way to create an ABC using inheritance.

__iter__()

__next__()

add(*components: cni.physicalComponent.PhysicalComponent*) → None

addToRegion(*region: pya.Region*, *filter:* ShapeFilter)

clone(*nameMap: cni.physicalComponent.NameMapper = NameMapper()*, *netMap: cni.physicalComponent.NameMapper = NameMapper()*)

destroy()

getComps() → list

getComp(*index: int*) → cni.physicalComponent.PhysicalComponent

moveBy(*dx: float*, *dy: float*) → None

toString()

transform(*transform: cni.physicalComponent.Transform*) → None

4.5 shape

4.5.1 Module Contents

Classes

Shape	Helper class that provides a standard way to create an ABC using

class shape.**Shape**(*layer: cni.shapefilter.Layer*, *bbox: cni.box.Box*)

Bases: cni.physicalComponent.PhysicalComponent

Helper class that provides a standard way to create an ABC using inheritance.

property bbox

The bounding box for this Shape

property layer

The Layer associated with this Shape

classmethod getCell() → pya.Cell

set_shape(*shape:* Shape)

getShape()

getBBox(*filter: cni.shapefilter.ShapeFilter = ShapeFilter()*) → cni.box.Box

getLayer() → cni.shapefilter.Layer

getNet() → *Net*

getPin() → *Pin*

setPin(*pin:* Pin) → None

4.6 dlo

4.6.1 Module Contents

Classes

ChoiceConstraint	Built-in mutable sequence.
RangeConstraint	
PyCellContext	
PCellWrapper	

class dlo.**ChoiceConstraint**(*choices*, *action=REJECT*)

Bases: list

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

class dlo.**RangeConstraint**(*low*, *high*, *resolution=None*, *action=REJECT*)

class dlo.**PyCellContext**(*tech*, *cell*, *impl*)

Bases: object

property cell

property tech

property layout

property impl

property cell

classmethod getCurrentPyCellContext() → *PyCellContext*

__enter__()

__exit__(**params*)

class dlo.**PCellWrapper**(*impl*, *tech*)

Bases: pya.PCellDeclaration

__call__(*name*, *value*, *description=None*, *constraint=None*)

get_parameters()

params_as_hash(*parameters*)

display_text(*parameters*)

produce(*layout*, *layers*, *parameters*, *cell*)

4.7 path

4.7.1 Module Contents

Classes

Path	Creates a path object, using the points list of points to define the path. The width

class path.**Path**(*arg1*, *arg2*, *arg3*, *arg4=None*)

Bases: cni.shape.Shape

Creates a path object, using the points list of points to define the path. The width parameter is used to specify the width of this path. The layer parameter is a Layer object, the width parameter is an integer value, and the points parameter is a Python list of Point objects.

Parameters

- **layer** (Layer) – Layer to place the path
- **width** (*float*) – width of the path
- **points** (PointList) – pointlist to define the path

addToRegion(*region: pya.Region*, *filter: cni.shape.ShapeFilter*)

clone(*nameMap: cni.shape.NameMapper = NameMapper()*, *netMap: cni.shape.NameMapper = NameMapper()*)

destroy()

getPoints() → cni.pointlist.PointList

moveBy(*dx: float*, *dy: float*) → None

toString() → str

transform(*transform: cni.shape.Transform*) → None

4.8 paramarray

4.8.1 Module Contents

Classes

ParamArray	dict() -> new empty dictionary

class paramarray.**ParamArray**(**arg*, ***kw*)

Bases: dict

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:
d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs**
in the keyword argument list. For example: dict(one=1, two=2)

4.9 ulist

4.9.1 Module Contents

Classes

ulist	Built-in mutable sequence.

Attributes

T

ulist.**T**

class ulist.**ulist**(*items=None*)

Bases: list[*T*]

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

append(*item*) → None

Append object to the end of the list.

4.10 location

4.10.1 Module Contents

Classes

Location

class location.**Location**

Bases: object

LOWER_LEFT = 1

CENTER_LEFT = 2

UPPER_LEFT = 3

LOWER_CENTER = 4

CENTER_CENTER = 5

UPPER_CENTER = 6

LOWER_RIGHT = 7

CENTER_RIGHT = 8

UPPER_RIGHT = 9

mirrorX()

mirrorY()

rotate90()

rotate180()

rotate270()

transform(*transform*)

4.11 termtype

4.11.1 Module Contents

Classes

TermType

class termtype.**TermType**

Bases: object

INPUT = 1

OUTPUT = 2

INPUT_OUTPUT = 3

SWITCH = 4

JUMPER = 5

UNUSED = 6

TRISTATE = 7

4.12 point

4.12.1 Module Contents

Classes

Point

class point.**Point**(*x*, *y*)

Bases: object

property x

Returns the value of the x-coordinate for this point

property y

Returns the value of the y-coordinate for this point

classmethod areColinearPoints(*p1*, *p2*, *p3*)

Returns True if these three points are colinear or coincident, and returns False otherwise.

Parameters

- **p1** (Point) – first point.
- **p2** (Point) – second point.
- **p3** (Point) – third point.

Returns
whether all three points are collinear or coincident

Return type
boolean

copy()

getCoord(*dir*)

getSpacing(*dir*, *refPoint*)

getX()

getY()

invalid()

isBetween(*a*, *b*)

isValid()

place(*dir*, *refPoint*, *distance*, *align=True*)

set(*p*)

set(*_x*, *_y*)

setCoord(*dir*, *coord*)

setX(*x*)

setY(*y*)

snap(*grid*, *snapType=None*)

snapX(*grid*, *snapType=None*)

snapY(*grid*, *snapType=None*)

snapTowards(*grid*, *dir*)

toDiagAxes()

toOrthogAxes()

transform(*trans*)

__eq__(*other*)
Return self==value.

4.13 tech

4.13.1 Module Contents

Classes

TechImpl
Tech

class tech.**TechImpl**
Bases: object

class tech.**Tech**
Bases: object

techsByName

techInUse =

register()

get()

4.14 orientation

4.14.1 Module Contents

Classes

Orientation

class orientation.**Orientation**

Bases: object

R0 = 0

R90 = 1

R180 = 2

R270 = 3

MY = 4

MYR90 = 5

MX = 6

MXR90 = 7

concat(*other*)

getRelativeOrient(*other*)

4.15 constants

4.15.1 Module Contents

constants.**REJECT = 1**

constants.**ACCEPT = 2**

constants.**USE_DEFAULT = 3**

constants.**INT_MAX**

constants.**INT_MIN**

4.16 pathstyle

4.16.1 Module Contents

Classes

PathStyle	

class pathstyle.**PathStyle**

Bases: object

TRUNCATE = 1

EXTEND = 2

ROUND = 3

VARIABLE = 4

4.17 pointlist

4.17.1 Module Contents

Classes

PointList	Built-in mutable sequence.

class pointlist.**PointList**(*items=None*)

Bases: cni.ulist.ulist[cni.point.Point]

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

compress(*isClose=True*) → *PointList*

Compresses this PointList, by removing any extra (coincident and/or collinear) points from this PointList. The optional isClosed parameter is used to indicate whether this set of points is meant to represent a closed shape or not. If all points are collinear, then the first and last points will be the result of compressing this PointList. If the first and last points are coincident, then only the first point is returned

Parameters
isClose (*boolean*) – Whether represented shape is closed

Returns
see description above

Return type
PointList

containsPoint(*point: cni.point.Point*) → bool

4.18 dlogen

4.18.1 Module Contents

Classes

Dlo
DloGen

class dlogen.**Dlo**(*libName*, *cellName*, *viewName='layout'*, *viewType=None*)

Bases: object

classmethod exists(*dloName: str*) → bool

Returns True if the dloName Dlo design object exists, and False otherwise. The dloName is a string of the form "<libName>/<cellName>/<viewName>". If <viewName> is not specified, then the default value "layout" will be used.

Parameters
dloName (*str*) – name of dlo object

Returns
wether cell exists

Return type
bool

findPin(*name: str*) → cni.pin.Pin

hasPin(*name: str*) → bool

hasTerm(*name: str*) → bool

hasNet(*name: str*) → bool

findTerm(*name: str*) → cni.term.Term

addCellContext(*cell*)

class dlogen.**DloGen**

Bases: *Dlo*

classmethod setLibName(*libName: str*) → None

classmethod getLibName() → str

setTech(*tech*)

addPin(*pinName: str*, *termName: str*, *box: cni.pin.Box*, *layer: cni.pin.Layer*) → cni.pin.Pin

4.19 transform

4.19.1 Module Contents

Classes

Transform	The Transform class provides the ability to implement two-dimensional

class transform.**Transform**(*arg1*, *arg2*, *arg3*, *arg4=None*)

Bases: object

The Transform class provides the ability to implement two-dimensional transformations, consisting of orientation changes (rotations and mirroring about the coordinate axes), translation (offsets in the X and Y directions), and magnification of the X and Y coordinates, with the operations performed in the following order:

1. Rotation/Mirroring
2. Translation
3. Magnification

When rotation operations are performed on an object in the layout design, it is important to note that it may be necessary to first translate the object to the origin of the DLO coordinate system, apply the rotation operation, and then translate the object back to its original location. This would be necessary, because the center of rotation is the origin of the coordinate system, not the center of the object. With this approach, the object will be rotated about the center of the object. Otherwise, the resulting rotation may not produce the expected results. In order to more easily handle this situation, the rotate() methods are provided by this Transform class.

Creation:

The Transform object can be directly created using the desired x and y coordinate values for the translation operation, the desired orientation value, and the desired magnification value. The individual x and y coordinate values can be specified, or the corresponding Point object can be used instead. Thus, this Transform object can be created using either of the following forms:

Transform(Coord x, Coord y, Orientation o=R0, double mag=1.0)

Transform(Point offset, Orientation o=R0, double mag=1.0)

If these values are not specified, then default values will be generated and used.

property transform

returns the internal transform representation

property xOffset

returns the x-coordinate value of the offset for this Transform

property yOffset

returns the y-coordinate value of the offset for this Transform

property mag

returns the magnification value for this Transform

property orientation

returns the orientation value for this Transform

4.20 rect

4.20.1 Module Contents

Classes

Rect	Helper class that provides a standard way to create an ABC using

class rect.**Rect**(*layer: cni.layer.Layer*, *box: cni.box.Box*)

Bases: cni.shape.Shape

Helper class that provides a standard way to create an ABC using inheritance.

property bottom

property left

property right

property top

addToRegion(*region: cni.shape.pya.Region*, *filter: cni.shape.ShapeFilter*)

clone(*nameMap: cni.shape.NameMapper = NameMapper()*, *netMap: cni.shape.NameMapper = NameMapper()*)

destroy()

moveBy(*dx: float*, *dy: float*) → None

toString() → str

transform(*transform: cni.shape.Transform*) → None

4.21 geo

4.21.1 Module Contents

Functions

fgOr(→ cni.grouping.Grouping)	Performs a logical OR operation for lists of physical components components1 and components2, by
fgAnd(→ cni.grouping.Grouping)	Performs a logical AND operation for lists of physical components components1 and components2,
fgXor(→ cni.grouping.Grouping)	Performs a logical XOR operation for lists of physical components components1 and components2,
fgNot(→ cni.grouping.Grouping)	Performs a logical NOT operation for lists of physical components components1 and components2,
fgSize(→ cni.grouping.Grouping)	Expands or shrinks all polygon areas contained in this list of physical components, according to

geo.**fgOr**(*components1: cni.grouping.ulist[cni.grouping.PhysicalComponent]*, *components2: cni.grouping.ulist[cni.grouping.PhysicalComponent]*, *resultLayer:* Layer) → cni.grouping.Grouping

Performs a logical OR operation for lists of physical components components1 and components2, by selecting those polygon areas which are in either list of physical components. The resulting merged polygon shapes are generated on the resultLayer layer. In addition, these polygon shapes are used to create a Grouping object, which is the return value for this method.

Parameters

- **components1** (*list of PhysicalCompent*) – first list of physical component derived objects
- **components2** (*list of PhysicalCompent*) – second list of physical component derived objects
- **resultLayer** (Layer) – layer where resulting shapes will be generated on

Returns
grouping object

Return type
Grouping

geo.**fgAnd**(*components1: cni.grouping.ulist[cni.grouping.PhysicalComponent]*, *components2: cni.grouping.ulist[cni.grouping.PhysicalComponent]*, *resultLayer:* Layer) → cni.grouping.Grouping

Performs a logical AND operation for lists of physical components components1 and components2, by selecting those polygon areas which are in both physical components. The resulting polygon shapes are generated on the resultLayer layer. In addition,these polygon shapes are used to create a Grouping object, which is the return value for this method. If there are no polygon shapes, then this Grouping object is empty.

Parameters

- **components1** (*list of PhysicalCompent*) – first list of physical component derived objects
- **components2** (*list of PhysicalCompent*) – second list of physical component derived objects
- **resultLayer** (Layer) – layer where resulting shapes will be generated on

Returns
grouping object

Return type
Grouping

geo.**fgXor**(*components1: cni.grouping.ulist[cni.grouping.PhysicalComponent]*, *components2: cni.grouping.ulist[cni.grouping.PhysicalComponent]*, *resultLayer:* Layer) → cni.grouping.Grouping

Performs a logical XOR operation for lists of physical components components1 and components2, by selecting those polygon areas which are in either list of physical components, but not in both lists of physical components. The resulting merged polygon shapes are generated on the resultLayer layer. In addition, these polygon shapes are used to create a Grouping object, which is the return value for this method.

Parameters

- **components1** (*list of PhysicalCompent*) – first list of physical component derived objects
- **components2** (*list of PhysicalCompent*) – second list of physical component derived objects
- **resultLayer** (Layer) – layer where resulting shapes will be generated on

Returns
grouping object

Return type
Grouping

geo.**fgNot**(*components1: cni.grouping.ulist[cni.grouping.PhysicalComponent]*, *components2: cni.grouping.ulist[cni.grouping.PhysicalComponent]*, *resultLayer:* Layer) → cni.grouping.Grouping

Performs a logical NOT operation for lists of physical components components1 and components2, by selecting those polygon areas contained in the components1 physical component which are not contained in the components2 physical component. The resulting polygon shapes are generated on the resultLayer layer. In addition, these polygon shapes are used to create a Grouping object, which is the return value for this method. If there are no polygon shapes generated, then this Grouping object will be empty.

Parameters

- **components1** (*list of PhysicalCompent*) – first list of physical component derived objects
- **components2** (*list of PhysicalCompent*) – second list of physical component derived objects
- **resultLayer** (Layer) – layer where resulting shapes will be generated on

Returns
grouping object

Return type
Grouping

geo.**fgSize**(*components: cni.grouping.ulist[cni.grouping.PhysicalComponent]*, *filter: cni.shapefilter.ShapeFilter*, *sizeValue: float*, *resultLayer:* Layer, *grid: float = None*) → cni.grouping.Grouping

Expands or shrinks all polygon areas contained in this list of physical components, according to the sizeValue parameter value. If this sizeValue parameter is positive, then the polygon edges are expanded outward by that amount. If this sizeValue parameter is negative, then the polygon edges are shrunk inward by that amount. The resulting polygon shapes are generated on the resultLayer layer. In addition, these polygon shapes are used to create a Grouping object, which is the return value for this method. Note that filter shape filter is first used to merge all shapes from the comps list, and then the SIZE geometrical operation is performed on these merged geometries. If the grid parameter is specified, then the points of the physical components are snapped with grid value to the nearest grid point.

Parameters

- **components** (*list of PhysicalCompent*) – list of physical component derived objects
- **filter** (ShapeFilter) – layer filter to use
- **sizeValue** (*float*) – value for sizing (bias)
- **resultLayer** (Layer) – layer where resulting shapes will be generated on
- **grid** (*float*) – optional value for snapping

Returns
grouping object

Return type
Grouping

4.22 shapefilter

4.22.1 Module Contents

Classes

ShapeFilter	Creates an empty ShapeFilter object, which would be used to indicate that all layers should be

class shapefilter.**ShapeFilter**(*arg=None*)

Bases: object

Creates an empty ShapeFilter object, which would be used to indicate that all layers should be considered in any bounding box or placement calculations

Creates a ShapeFilter object, consisting of only a single layer. This single layer is the only layer which should be considered when the ShapeFilter object is passed to methods which perform bounding box and placement calculations.

Parameters
layer (Layer) – Layer to filter

Creates a ShapeFilter object, consisting of a list of Layer objects. These layers are the only layers which should be considered when the ShapeFilter is passed to methods which perform bounding box or placement calculations.

Parameters
layerList (*LayerList*) – Layers to filter

Creates a ShapeFilter object, consisting of the Layer objects which are specified by the shapeFilter ShapeFilter object. These layers are the only layers which should be considered when the ShapeFilter is passed to methods which perform bounding box or placement calculations.

Parameters
shapeFilter (ShapeFilter) – ShapeFilter to filter

isIncluded(*layer: cni.layer.Layer*) → bool

Returns true if the layer parameter is a layer which is in the list of layers considered by the ShapeFilter object, and returns False otherwise.

Parameters
layer (Layer) – Layer to check

Return type
bool

4.23 polygon

4.23.1 Module Contents

Classes

Polygon	Helper class that provides a standard way to create an ABC using

class polygon.**Polygon**(*arg1*, *arg2*, *arg3=None*)

Bases: cni.shape.Shape

Helper class that provides a standard way to create an ABC using inheritance.

addToRegion(*region: pya.Region*, *filter: cni.shape.ShapeFilter*)

clone(*nameMap: cni.shape.NameMapper = NameMapper()*, *netMap: cni.shape.NameMapper = NameMapper()*)

destroy()

getPoints() → cni.pointlist.PointList

moveBy(*dx: float*, *dy: float*) → None

toString() → str

transform(*transform: cni.shape.Transform*) → None

4.24 numeric

4.24.1 Module Contents

Classes

Numeric	The Numeric class is used to create a floating point number from a string

class numeric.**Numeric**

Bases: float

The Numeric class is used to create a floating point number from a string representation, such as "10ns". This string representation is composed of two parts: 1) a number part and 2) a scale factor part. Thus, this Numeric class can be used to represent a floating point number as a floating point number along with a scaling factor. Since this Numeric class is derived from the base Python float class, it can be used just like a regular floating point number in any numerical computation.

The number part of this Numeric class string representation can be any valid Python integer or floating point number; this Python floating point number can be represented using standard scientific notation, such as "1.23e-4". The scaling factor part of this Numeric class string representation must be one of the following pre-defined scaling factor string values:

Character	Name	Multiplier
Y	Yotta	1e24
Z	Zetta	1e21
E	Exa	1e18
P	Peta	1e15
T	Tera	1e12
G	Giga	1e09
M	Mega	1e06
K or k	Kilo	1e03
‘’	no scale factor	1.0
%	percent	1e-2
c	centi	1e-2
m	milli	1e-3
u	micron	1e-6
n	nano	1e-9
p	pico	1e-12
f	femto	1e-15
a	atto	1e-18
z	zepto	1e-21
y	yocto	1e-24

Note that any characters after the first character in the scaling factor are simply ignored. Thus, the scaling factor “mVolt” is the same as “m”. This capability can be used to create more descriptive scaling factors.

Numeric(int | float | string) – creates a Numeric object, based upon the specified number or string. The string must be a string of the form <number><scaleFactor>, where the <scaleFactor> is one of the pre-defined scaling factors in the above table of scaling factor strings. That is, this string representation must be composed of a number part and a scaling factor part, where the scaling factor is a pre-defined scaling factor string.

property scaleFactor

The default (original) scale factor

property scale_factors

List of all available scaling factors, along with their values

scaleFormat(*scaleFactor=None*)

Returns the floating point number formatted using the specified scaleFactor scaling value. If this scaleFactor parameter is not specified, then the floating point number is returned using the scale factor which was used when the Numeric class object was created.

Parameters
scaleFactor (*string or None*) – Optional scaling factor to use.

Returns
new scaled Numeric object

Return type
Numeric

4.25 instance

4.25.1 Module Contents

Classes

Instance	Creates an Instance object, where the dloName parameter specifies the Klayout name to be

class instance.**Instance**(*dloName: str*)

Creates an Instance object, where the dloName parameter specifies the Klayout name to be used for this Instance object. The dloName parameter is a string of the form "libName/cellName/viewName", where the libName and the viewName are optional. If the libName is not specified, then the library name associated with the current DloGen is used; if the viewName is not specified, then the default viewName is used (currently "layout").

Parameters
dloName (*str*) – name of dlo object

getParams() → cni.paramarray.ParamArray

Returns the ParamArray which provides the explicit parameters and values which were used when this Instance object was created.

Returns
array of parameters

Return type
ParamArray

setParams(*params: cni.paramarray.ParamArray*) → None

Uses the passed ParamArray params to set the parameter values for this Instance.

Parameters
params (ParamArray) – parameters to set

setOrientation(*orientation: cni.orientation.Orientation*) → None

Sets the orientation for this Instance.

Parameters
orientation (Orientation) – orientation to set

setOrigin(*point: cni.shape.Point*) → None

Sets the Point point parameter to be the origin for this Instance.

Parameters
point (Point) – origin to set

4.26 signaltype

4.26.1 Module Contents

Classes

SignalType	

class signaltype.**SignalType**

Bases: object

SIGNAL = 1

POWER = 2

GROUND = 3

CLOCK = 4

TIEOFF = 5

TIEHI = 6

TIELO = 7

ANALOG = 8

SCAN = 9

RESET = 10

4.27 text

4.27.1 Module Contents

Classes

Text	Helper class that provides a standard way to create an ABC using

class text.**Text**(*layer: cni.rect.Layer*, *text: str*, *origin: cni.rect.Point*, *height: float*)

Bases: cni.rect.Shape

Helper class that provides a standard way to create an ABC using inheritance.

addToRegion(*region: pya.Region*, *filter: cni.rect.ShapeFilter*)

clone(*nameMap: cni.rect.NameMapper = NameMapper()*, *netMap: cni.rect.NameMapper = NameMapper()*)

destroy()

moveBy(*dx: float*, *dy: float*) → None

setAlignment(*location: cni.location.Location*) → None

setOrientation(*orient*)

setDrafting(*drafting*)

transform(*transform: cni.transform.Transform*) → None

4.28 layer

4.28.1 Module Contents

Classes

Layer

class layer.**Layer**(*name*, *purpose=None*)

Bases: object

property name

property number

property purposeName

property purposeNumber

tech

layout

getAttrs()

getGridResolution()

getLayerAbove()

getLayerAbove(*layerMaterial*)

getLayerBelow()

getLayerBelow(*layerMaterial*)

getLayerName()

getLayerNumber()

getMaterial()

getPurposeName()

getPurposeNumber()

getRoutingDir()

isAbove(*layer*)

isMaskLayer()

4.29 term

4.29.1 Module Contents

Classes

Term	Creates a Term terminal object in the current design, using the termName parameter to name this

class term.**Term**(*termName: str*, *termType: cni.termtype.TermType = TermType.INPUT_OUTPUT*)

Bases: object

Creates a Term terminal object in the current design, using the termName parameter to name this newly created Term object. If there is already a net in the current design having the same name as this terminal, then this net will be connected to this terminal. Otherwise, a new net will be created, and connected to this terminal, using this termName string to name this newly created net. If the termName string parameter is empty, or is the name of an existing terminal in the design, then an exception is raised. In addition, the termType parameter is used to specify the terminal type for this terminal.

Parameters

- **termName** (*string*) – name of the new terminal
- **termType** (*Termtype*) – optional type of the new terminal

property name

Returns the name of this terminal

addPin(*pin:* Pin) → None

getName() → str

Returns the name for this terminal.

Returns
The name of the terminal

Return type
string

getNet() → cni.net.Net

Returns the net associated with this Term terminal object. If there is no associated net, then an exception is raised.

Returns
The associated net

Return type
Net

getPins() → list[*Pin*]

Returns a uniform list of all of the Pin objects associated with this Term terminal object.

Returns
List of Pin

Return type
list[*Pin*]

setName(*name: str*) → None

Sets the name for this terminal. Note that the name of the associated net for this terminal is also changed to have the same name as this terminal. This is done in order to ensure that any net connected to this terminal will always have the same name as this terminal. If this name string parameter is empty, or is the name of an existing terminal or net in the design, then an exception is raised.

Parameters
name (*string*) – The name of the terminal

4.30 pin

4.30.1 Module Contents

Classes

Pin	Creates a Pin object for the specified terminal in the current design, using the pinName

class pin.**Pin**(*pinName: str*, *termName: str*, *shape: cni.shape.Shape = None*)

Bases: object

Creates a Pin object for the specified terminal in the current design, using the pinName parameter to name this newly created Pin object. If there is already a terminal in the current design having the same name as the termName parameter, then this terminal will be used to create this Pin object. Otherwise, a new terminal will be created, using this termName string to name this newly created terminal. If there is already a pin in the current design having the same name as the pinName string parameter, then an exception is raised. In addition, the shape parameter can optionally be used to associate a shape object with this pin.

Parameters

- **pinName** (*string*) – Name of the pin.
- **termName** (*string*) – second point.
- **shape** (Shape) – optional shape to associate

property name

Returns the name of this pin

addShape(*arg*)

getName() → str

Returns the name for this Pin object.

Returns
pin name

Return type
string

setBBox(*box: cni.box.Box*, *layer: cni.layer.Layer*) → None

getBBox() → cni.box.Box

setName(*name: str*) → None

Sets the name for this Pin object. If there is already a pin in the current design with the same name as the name parameter, an exception is raised.

Parameters
name (*string*) – Name to set

setTerm(*term: cni.term.Term*) → None

Associates the term terminal with this Pin object. Note that multiple Pin objects may be associated with a single terminal.

Parameters
Term – Terminal to associate

getTerm() → cni.term.Term

4.31 physicalComponent

4.31.1 Module Contents

Classes

PhysicalComponent	Helper class that provides a standard way to create an ABC using

class physicalComponent.**PhysicalComponent**

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

abstract addToRegion(*region: pya.Region*)

abstract clone(*nameMap: cni.namemapper.NameMapper = NameMapper()*, *netMap: cni.namemapper.NameMapper = NameMapper()*)

fgOr(*component:* PhysicalComponent, *resultLayer:* Layer) → *Grouping*

Performs a logical or operation for this physical component and another physical component, by selecting those polygon areas which are in either physical component. The resulting merged polygon shapes are generated on the resultLayer layer. In addition, these polygon shapes are used to create a Grouping object, which is the return value for this method.

Parameters

- **component** (*PhysicalCompent*) – physical component derived object
- **resultLayer** (Layer) – layer where resulting shapes will be generated on

Returns
grouping object

Return type
Grouping

fgXor(*component:* PhysicalComponent, *resultLayer:* Layer) → *Grouping*

Performs a logical xor operation for this physical component and another physical component, by selecting those polygon areas which are in either physical component, but not in both lists of physical components. The resulting merged polygon shapes are generated on the resultLayer layer. In addition, these polygon shapes are used to create a Grouping object, which is the return value for this method.

Parameters

- **component** (*PhysicalCompent*) – physical component derived object
- **resultLayer** (Layer) – layer where resulting shapes will be generated on

Returns
grouping object

Return type
Grouping

fgAnd(*component:* PhysicalComponent, *resultLayer:* Layer) → *Grouping*

Performs a logical and operation for this physical component and the component physical component, by selecting those polygon areas which are in both physical components. The resulting polygon shapes are generated on the resultLayer layer. In addition, these polygon shapes are used to create a Grouping object, which is the return value for this method. If there are no polygon shapes, then this Grouping object is empty.

Parameters

- **component** (*PhysicalCompent*) – physical component derived object
- **resultLayer** (Layer) – layer where resulting shapes will be generated on

Returns
grouping object

Return type
Grouping

fgNot(*component:* PhysicalComponent, *resultLayer:* Layer) → *Grouping*

Performs a logical not operation for this physical component and another physical component, by selecting those polygon areas contained in this physical component which are not contained in the component physical component. The resulting polygon shapes are generated on the resultLayer layer. In addition, these polygon shapes are used to create a Grouping object, which is the return value for this method. If there are no polygon shapes generated, then this Grouping object will be empty.

Parameters

- **component** (*PhysicalCompent*) – physical component derived object
- **resultLayer** (Layer) – layer where resulting shapes will be generated on

Returns
grouping object

Return type
Grouping

fgSize(*filter:* ShapeFilter, *sizeValue: float*, *resultLayer:* Layer, *grid: float = None*) → *Grouping*

Expands or shrinks all polygon areas contained in this physical component, according to the sizeValue parameter value. If this sizeValue parameter is positive, then the polygon edges are expanded outward by that amount. If this sizeValue parameter is negative, then the polygon edges are shrunk inward by that

amount. The resulting polygon shapes are generated on the resultLayer layer. In addition, these polygon shapes are used to create a Grouping object, which is the return value for this method. If the grid parameter is specified, then the points of the physical components are snapped with grid value to the nearest grid point.

Parameters

- **filter** (ShapeFilter) – layer filter to use
- **sizeValue** (*float*) – value for sizing (bias)
- **resultLayer** (Layer) – layer where resulting shapes will be generated on
- **grid** (*float*) – optional value for snapping

Returns
grouping object

Return type
Grouping

abstract destroy()

abstract moveBy(*dx: float*, *dy: float*) → None

abstract transform(*transform: cni.transform.Transform*) → None

4.32 net

4.32.1 Module Contents

Classes

Net	Creates a Net object in the current design, using the netName parameter to name this newly

class net.**Net**(*netName: str*, *sigType: cni.signaltype.SignalType = SignalType.SIGNAL*, *isGlobal: bool = False*)

Bases: object

Creates a Net object in the current design, using the netName parameter to name this newly created Net object. If this netName string parameter is empty, or is the name of anexisting net in the design, then an exception is raised. In addition, the sigType parameter is used to specify the signal type for this net, and the isGlobal Boolean flag parameter should be set to True, when this net is a global net for the current design.

Parameters

- **netName** (*string*) – name of the new net
- **sigType** (SignalType) – optional signal type of the new net

property name

Returns the name of this terminal

addTerm(*term:* Term) → None

getName() → str

Returns the name for this Net.

Returns
The name of the Net

Return type
string

getPins() → list[*Pin*]

Returns a list of pins which are connected to any terminal associated with this Net object. All Pin objects associated with this terminal are returned in this list.

Returns
List of Pin

Return type
list[*Pin*]

setName(*name: str*) → None

Sets the name for this Net object. Note that the name of any associated terminal for this Net will also be changed to use this new name. This is done in order to ensure that any terminal connected to a net will always have the same name as the net. If this name string parameter is empty, or is the name of an existing net or terminal in the current design, then an exception is raised.

Parameters
name (*string*) – The name of the net

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

O

P

R

S

T

U

V

X

Y