PyCell4Klayout

Release 0.1

IHP Authors

Feb 22, 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

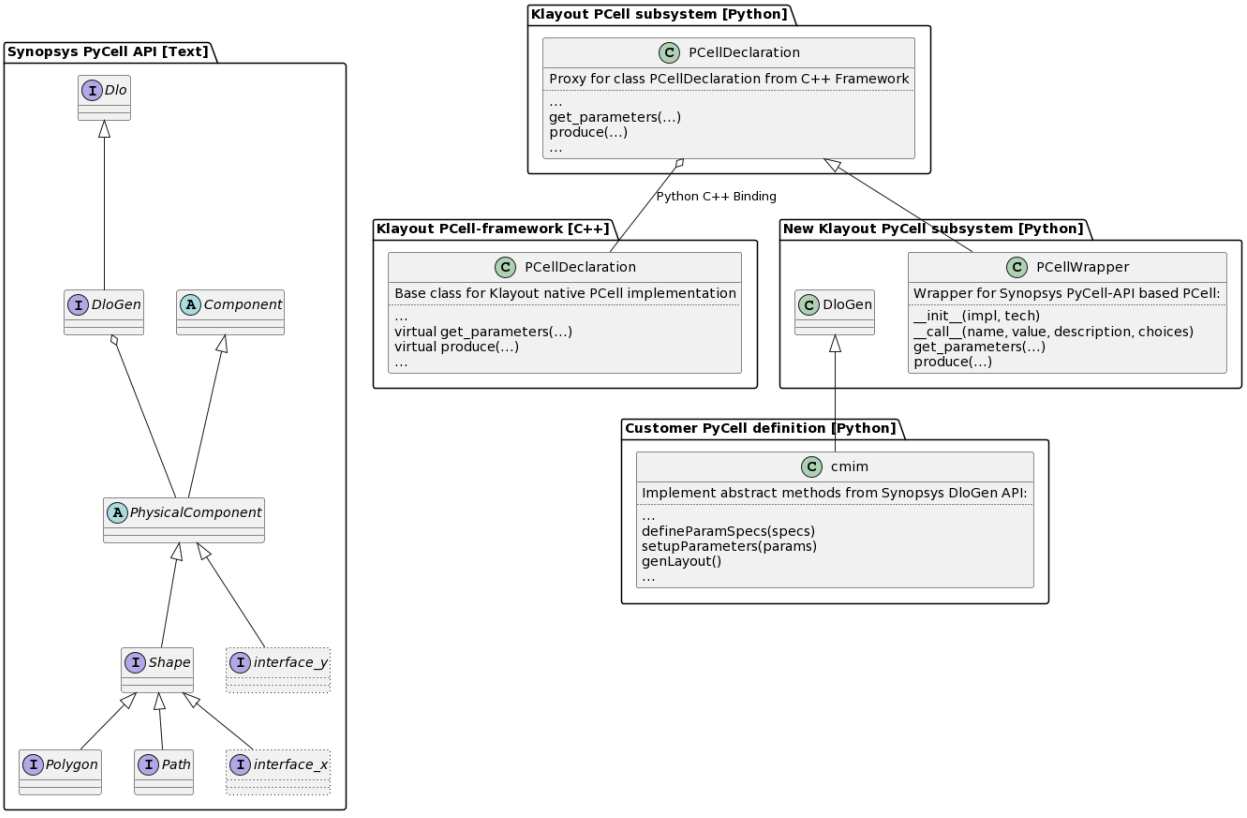

Fig. 1: PyCell wrapper structural architecture overview

CHAPTER

FOUR

API REFERENCE

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

4.1 cni

4.1.1 Submodules

cni.box

Module Contents

Classes

Box

class cni.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(*refBox*, *align=True*)

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

alignEdgeToCoord(*coord*)

alignEdgeToPoint(*point*)

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

alignLocationToPoint(*pt*)

[1] Created with sphinx-autoapi

centerCenter()

centerLeft()

centerRight()

contains(*incEdges=True*)

containsPoint(*incEdges=True*)

expand()

expandDir(*coord*)

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

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

expandToGrid(*dir=None*)

fix()

getArea()

getCenter()

getCenterX()

getCenterY()

getCoord()

getDimension()

getHeight()

getLeft()

getLocationPoint()

getLocationPoint()

getPoints()

getRange()

getRangeX()

getRangeY()

getRight()

getSpacing(*refBox*)

getTop()

getWidth()

hasNoArea()

init()

intersect()

intersect(*dir*)

isInverted()

isNormal()

limit()

lowerCenter()

lowerLeft()

lowerRight()

merge(*dir*)

mergePoint()

mirrorX()

mirrorY()

moveBy(*dy*)

moveTo(*loc=Location.CENTER_CENTER*)

moveTowards(*d*)

overlaps(*incEdges=True*)

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

removeRegion()

rotate90()

rotate180()

rotate270()

set()

set(*dir=None*)

set(*upperRight*)

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

setBottom()

setCenter()

setCenterY()

setCoord(*coord*)

setDimension(*dir*)

setBottom()

setHeight()

setLocationPoint(*pt*)

setRange(*range*)

setRangeX()

setRangeY()

setRight()

setTop()

setWidth()

snap(*snapType=None*)

snapX(*snapType=None*)

snapY(*snapType=None*)

snapTowards(*dir*)

transform()

upperCenter()

upperLeft()

upperRight()

cni.constants

Module Contents

cni.constants.**REJECT = 1**

cni.constants.**ACCEPT = 2**

cni.constants.**USE_DEFAULT = 3**

cni.constants.**INT_MAX**

cni.constants.**INT_MIN**

cni.dlo

Module Contents

Classes

TechImpl	
Tech	
PyCellContext	
PCellWrapper	

Functions

ChoiceConstraint(choices[, action])	

class cni.dlo.**TechImpl**

Bases: object

class cni.dlo.**Tech**

Bases: object

techsByName

register()

get()

cni.dlo.**ChoiceConstraint**(*choices*, *action=ACCEPT*)

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

Bases: object

__enter__()

__exit__(**params*)

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

Bases: pya.PCellDeclaration

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

get_parameters()

params_as_hash(*parameters*)

display_text(*parameters*)

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

cni.dlogen

Module Contents

Classes

DloGen

class cni.dlogen.**DloGen**

Bases: object

set_tech(*tech*)

addPin(*name*, *label*, *box*, *layer*)

cni.font

Module Contents

Classes

Font

class cni.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*)

cni.geo

Module Contents

Functions

fgOr()
fgAnd()
fgXor()
fgNot()
fgMerge()

cni.geo.**fgOr**()

cni.geo.**fgAnd**()

cni.geo.**fgXor**()

cni.geo.**fgNot**()

cni.geo.**fgMerge**()

cni.layer

Module Contents

Classes

Layer

class cni.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()

cni.location

Module Contents

Classes

Location

class cni.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*)

cni.numeric

Module Contents

Classes

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

class cni.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

cni.orientation

Module Contents

Classes

Orientation

class cni.orientation.**Orientation**

Bases: object

R0 = 0

R90 = 1

R180 = 2

R270 = 3

MY = 4

MYR90 = 5

MX = 6

MXR90 = 7

concat(*other*)

getRelativeOrient(*other*)

cni.pathstyle

Module Contents

Classes

PathStyle

class cni.pathstyle.**PathStyle**

Bases: object

TRUNCATE = 1

EXTEND = 2

ROUND = 3

VARIABLE = 4

cni.point

Module Contents

Classes

Point

class cni.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.

cni.pointlist

Module Contents

Classes

PointList	Built-in mutable sequence.

class cni.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 – Whether represented shape is closed

Returns

see description above

Return type

PointList

cni.polygon

Module Contents

Classes

Polygon

class cni.polygon.**Polygon**(*layer:* cni.layer.Layer, *pointList:* cni.pointlist.PointList)

Bases: *cni.shape.Shape*

cni.rect

Module Contents

Classes

Rect

class cni.rect.**Rect**(*layer*, *box*)

Bases: *cni.shape.Shape*

cni.shape

Module Contents

Classes

Shape

class cni.shape.**Shape**(*bbox=None*)

Bases: `object`

cell

set_shape(*sh*)

getBBox()

cni.signaltype

Module Contents

Classes

SignalType

class cni.signaltype.**SignalType**

Bases: object

SIGNAL = 1

POWER = 2

GROUND = 3

CLOCK = 4

TIEOFF = 5

TIEHI = 6

TIELO = 7

ANALOG = 8

SCAN = 9

RESET = 10

cni.termtype

Module Contents

Classes

TermType

class cni.termtype.**TermType**

Bases: object

INPUT = 1

OUTPUT = 2

INPUT_OUTPUT = 3

SWITCH = 4

JUMPER = 5

UNUSED = 6

TRISTATE = 7

cni.text

Module Contents

Classes

Text	

class cni.text.**Text**(*layer*, *text*, *point*, *size*)

Bases: cni.rect.Shape

setAlignment(*align*)

setOrientation(*orient*)

setDrafting(*drafting*)

cni.ulist

Module Contents

Classes

ulist	Built-in mutable sequence.

Attributes

T

cni.ulist.**T**

class cni.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() → None

Append object to the end of the list.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

N

O

P

R

S

T

U

V

X

Y