PyCell4Klayout

Release 0.1

IHP Authors

May 15, 2024

CONTENTS:

1 Referenced documents **2**

2 Introduction **3**
- 2.1 Basic Structure 3

3 Implementation Details **4**
- 3.1 Klayout 4
- 3.2 Synopsys PyCell API 4
- 3.3 Wrapper solution approach 5
- 3.4 PDK technology information handling 6

4 API Reference **7**
- 4.1 `font` 7
- 4.2 `namemapper` 8
- 4.3 `box` 8
- 4.4 `grouping` 11
- 4.5 `shape` 12
- 4.6 `dlo` 12
- 4.7 `paramarray` 13
- 4.8 `ulist` 14
- 4.9 `location` 14
- 4.10 `termtype` 15
- 4.11 `point` 16
- 4.12 `tech` 17
- 4.13 `orientation` 18
- 4.14 `constants` 18
- 4.15 `pathstyle` 19
- 4.16 `pointlist` 19
- 4.17 `dlogen` 20
- 4.18 `transform` 20
- 4.19 `rect` 21
- 4.20 `geo` 22
- 4.21 `polygon` 23
- 4.22 `numeric` 24
- 4.23 `instance` 25
- 4.24 `signaltype` 26
- 4.25 `text` 26
- 4.26 `layer` 27
- 4.27 `physicalComponent` 28

5 Indices and tables **30**

Python Module Index **31**

Index **32**

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

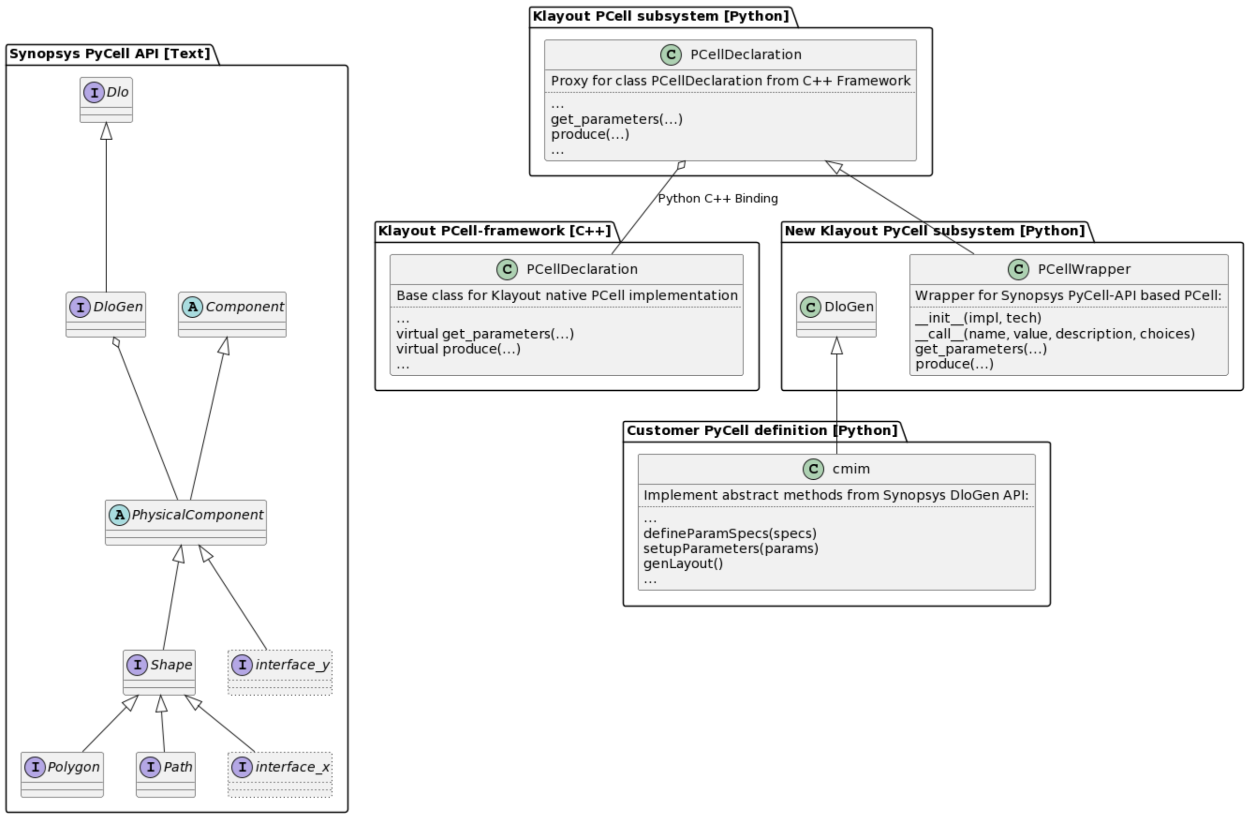
Synopsys PyCell API [Text]
Dlo
DloGen
Component
PhysicalComponent
Shape
interface_y
Polygon
Path
interface_x
Klayout PCell subsystem [Python]
PCellDeclaration
Proxy for class PCellDeclaration from C++ Framework
...
get_parameters(...)
produce(...)
...
Python C++ Binding
Klayout PCell-framework [C++]
PCellDeclaration
Base class for Klayout native PCell implementation
...
virtual get_parameters(...)
virtual produce(...)
...
New Klayout PyCell subsystem [Python]
DloGen
PCellWrapper
Wrapper for Synopsys PyCell-API based PCell:
__init__(impl, tech)
__call__(name, value, description, choices)
get_parameters(...)
produce(...)
Customer PyCell definition [Python]
cmim
Implement abstract methods from Synopsys DloGen API:
...
defineParamSpecs(specs)
setupParameters(params)
genLayout()
...

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(*refBox*, *align=True*)

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

alignEdgeToCoord(*coord*)

alignEdgeToPoint(*point*)

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

alignLocationToPoint(*pt*)

centerCenter()

centerLeft()

centerRight()

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

contains(*incEdges=True*)

containsPoint(*incEdges=True*)

destroy()

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`(*dx: float*, *dy: float*) → None

`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()

setRect(*rect*)

setRight()

setTop()

setWidth()

snap(*snapType=None*)

snapX(*snapType=None*)

snapY(*snapType=None*)

snapTowards(*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.

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

addToRegion(*region: pya.Region*)

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**(*bbox=None*)

Bases: `cni.physicalComponent.PhysicalComponent`

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

classmethod getCell() → pya.Cell

set_shape(*shape:* Shape)

getShape()

getBBox()

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*)

Bases: `object`

property cell

property tech

property layout

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 paramarray

4.7.1 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)

4.8 `ulist`

4.8.1 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.

4.9 `location`

4.9.1 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*)

4.10 termtype

4.10.1 Module Contents

Classes

TermType	

class termtype.**TermType**

Bases: object

INPUT = 1

OUTPUT = 2

INPUT_OUTPUT = 3

SWITCH = 4

JUMPER = 5

UNUSED = 6

TRISTATE = 7

4.11 point

4.11.1 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.

4.12 tech

4.12.1 Module Contents

Classes

TechImpl
Tech

class tech.**TechImpl**
Bases: object

class tech.**Tech**
Bases: object

techsByName

techInUse =

register()

get()

4.13 orientation

4.13.1 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*)

4.14 constants

4.14.1 Module Contents

constants.**REJECT = 1**

constants.**ACCEPT = 2**

constants.**USE_DEFAULT = 3**

constants.**INT_MAX**

constants.**INT_MIN**

4.15 `pathstyle`

4.15.1 Module Contents

Classes

PathStyle	

class pathstyle.**PathStyle**

Bases: `object`

TRUNCATE = 1

EXTEND = 2

ROUND = 3

VARIABLE = 4

4.16 `pointlist`

4.16.1 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 – Whether represented shape is closed

Returns
see description above

Return type
PointList

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

4.17 `dlogen`

4.17.1 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

class dlogen.**DloGen**

Bases: *Dlo*

classmethod setLibName(*libName: str*) → None

classmethod getLibName() → str

setTech(*tech*)

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

4.18 `transform`

4.18.1 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

4.19 rect

4.19.1 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*)

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

4.20 geo

4.20.1 Module Contents

Functions

fgOr(→ cni.grouping.Grouping)	Performs a logical OR operation for lists of physical components comps1
fgAnd()	
fgXor()	
fgNot()	
fgMerge()	

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 comps1 and comps2, 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()`

`geo.fgXor()`

`geo.fgNot()`

`geo.fgMerge()`

4.21 polygon

4.21.1 Module Contents

Classes

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

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

Bases: `cni.shape.Shape`

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

addToRegion(*region: pya.Region*)

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

4.22 `numeric`

4.22.1 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

4.23 instance

4.23.1 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

4.24 signaltype

4.24.1 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

4.25 text

4.25.1 Module Contents

Classes

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

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

Bases: `cni.rect.Shape`

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

`addToRegion`(*region: pya.Region*)

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

`destroy`()

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

`setAlignment`(*align*)

`setOrientation`(*orient*)

`setDrafting`(*drafting*)

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

4.26 `layer`

4.26.1 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()

4.27 physicalComponent

4.27.1 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

abstract **destroy**()

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

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

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

box, 8

c

constants, 18

d

dlo, 12
dlogen, 20

f

font, 7

g

geo, 22
grouping, 11

i

instance, 25

l

layer, 27
location, 14

n

namemapper, 8
numeric, 24

o

orientation, 18

p

paramarray, 13
pathstyle, 19
physicalComponent, 28
point, 16
pointlist, 19
polygon, 23

r

rect, 21

s

shape, 12
signaltype, 26

t

tech, 17
termtype, 15
text, 26
transform, 20

u

ulist, 14

INDEX

Symbols

`__call__()` (*dlo.PCellWrapper method*), 13
`__enter__()` (*dlo.PyCellContext method*), 13
`__eq__()` (*point.Point method*), 17
`__exit__()` (*dlo.PyCellContext method*), 13

A

`abut()` (*box.Box method*), 8
`ACCEPT` (*in module constants*), 18
`add()` (*grouping.Grouping method*), 11
`addPin()` (*dlogen.DloGen method*), 20
`addToRegion()` (*grouping.Grouping method*), 11
`addToRegion()` (*physicalComponent.PhysicalComponent method*), 28
`addToRegion()` (*polygon.Polygon method*), 23
`addToRegion()` (*rect.Rect method*), 22
`addToRegion()` (*text.Text method*), 27
`alignEdge()` (*box.Box method*), 8
`alignEdgeToCoord()` (*box.Box method*), 8
`alignEdgeToPoint()` (*box.Box method*), 8
`alignLocation()` (*box.Box method*), 8
`alignLocationToPoint()` (*box.Box method*), 8
`ANALOG` (*signaltype.SignalType attribute*), 26
`append()` (*ulist.ulist method*), 14
`areColinearPoints()` (*point.Point class method*), 16

B

`bottom` (*box.Box property*), 8
`bottom` (*rect.Rect property*), 22
`box`
    `module`, 8
`Box` (*class in box*), 8

C

`calcBBox()` (*font.Font method*), 7
`cell` (*dlo.PyCellContext property*), 13
`CENTER_CENTER` (*location.Location attribute*), 14
`CENTER_LEFT` (*location.Location attribute*), 14
`CENTER_RIGHT` (*location.Location attribute*), 15
`centerCenter()` (*box.Box method*), 8
`centerLeft()` (*box.Box method*), 8
`centerRight()` (*box.Box method*), 8
`ChoiceConstraint` (*class in dlo*), 12
`CLOCK` (*signaltype.SignalType attribute*), 26
`clone()` (*box.Box method*), 8
`clone()` (*grouping.Grouping method*), 11
`clone()` (*physicalComponent.PhysicalComponent method*), 28
`clone()` (*polygon.Polygon method*), 23
`clone()` (*rect.Rect method*), 22
`clone()` (*text.Text method*), 27
`compress()` (*pointlist.PointList method*), 19
`concat()` (*orientation.Orientation method*), 18
`constants`
    `module`, 18
`contains()` (*box.Box method*), 9
`containsPoint()` (*box.Box method*), 9
`containsPoint()` (*pointlist.PointList method*), 19
`copy()` (*point.Point method*), 16

D

`destroy()` (*box.Box method*), 9
`destroy()` (*grouping.Grouping method*), 11
`destroy()` (*physicalComponent.PhysicalComponent method*), 28
`destroy()` (*polygon.Polygon method*), 23
`destroy()` (*rect.Rect method*), 22
`destroy()` (*text.Text method*), 27
`display_text()` (*dlo.PCellWrapper method*), 13
`dlo`
    `module`, 12
`Dlo` (*class in dlogen*), 20
`dlogen`
    `module`, 20
`DloGen` (*class in dlogen*), 20

E

`EURO_STYLE` (*font.Font attribute*), 7
`exists()` (*dlogen.Dlo class method*), 20
`expand()` (*box.Box method*), 9
`expandDir()` (*box.Box method*), 9
`expandForMinArea()` (*box.Box method*), 9
`expandForMinWidth()` (*box.Box method*), 9
`expandToGrid()` (*box.Box method*), 9

EXTEND (*pathstyle.PathStyle attribute*), 19

F

fgAnd() (*in module geo*), 23
fgMerge() (*in module geo*), 23
fgNot() (*in module geo*), 23
fgOr() (*in module geo*), 22
fgOr() (*physicalComponent.PhysicalComponent method*), 28
fgXor() (*in module geo*), 23
fix() (*box.Box method*), 9
FIXED (*font.Font attribute*), 7
font
    module, 7
Font (*class in font*), 7

G

geo
    module, 22
get() (*tech.Tech method*), 17
get_parameters() (*dlo.PCellWrapper method*), 13
getArea() (*box.Box method*), 9
getAttrs() (*layer.Layer method*), 27
getBBox() (*shape.Shape method*), 12
getCell() (*shape.Shape class method*), 12
getCenter() (*box.Box method*), 9
getCenterX() (*box.Box method*), 9
getCenterY() (*box.Box method*), 9
getComp() (*grouping.Grouping method*), 12
getComps() (*grouping.Grouping method*), 12
getCoord() (*box.Box method*), 9
getCoord() (*point.Point method*), 16
getCurrentPyCellContext() (*dlo.PyCellContext class method*), 13
getDimension() (*box.Box method*), 9
getGridResolution() (*layer.Layer method*), 27
getHeight() (*box.Box method*), 9
getLayerAbove() (*layer.Layer method*), 27
getLayerBelow() (*layer.Layer method*), 27
getLayerName() (*layer.Layer method*), 28
getLayerNumber() (*layer.Layer method*), 28
getLeft() (*box.Box method*), 9
getLibName() (*dlogen.DloGen class method*), 20
getLocationPoint() (*box.Box method*), 9
getMaterial() (*layer.Layer method*), 28
getMembers() (*font.Font class method*), 7
getParams() (*instance.Instance method*), 25
getPoints() (*box.Box method*), 9
getPoints() (*polygon.Polygon method*), 23
getPurposeName() (*layer.Layer method*), 28
getPurposeNumber() (*layer.Layer method*), 28
getRange() (*box.Box method*), 9
getRangeX() (*box.Box method*), 9
getRangeY() (*box.Box method*), 9
getRelativeOrient() (*orientation.Orientation method*), 18
getRight() (*box.Box method*), 9
getRoutingDir() (*layer.Layer method*), 28
getShape() (*shape.Shape method*), 12
getSpacing() (*box.Box method*), 9
getSpacing() (*point.Point method*), 16
getTop() (*box.Box method*), 9
getWidth() (*box.Box method*), 9
getX() (*point.Point method*), 16
getY() (*point.Point method*), 16
GOTHIC (*font.Font attribute*), 7
GROUND (*signaltype.SignalType attribute*), 26
grouping
    module, 11
Grouping (*class in grouping*), 11

H

hasNoArea() (*box.Box method*), 9

I

init() (*box.Box method*), 9
INPUT (*termtype.TermType attribute*), 15
INPUT_OUTPUT (*termtype.TermType attribute*), 15
instance
    module, 25
Instance (*class in instance*), 25
INT_MAX (*in module constants*), 18
INT_MIN (*in module constants*), 18
intersect() (*box.Box method*), 9, 10
invalid() (*point.Point method*), 16
isAbove() (*layer.Layer method*), 28
isBetween() (*point.Point method*), 16
isInverted() (*box.Box method*), 10
isMaskLayer() (*layer.Layer method*), 28
isNormal() (*box.Box method*), 10
isValid() (*point.Point method*), 16

J

JUMPER (*termtype.TermType attribute*), 15

L

layer
    module, 27
Layer (*class in layer*), 27
layout (*dlo.PyCellContext property*), 13
layout (*layer.Layer attribute*), 27
left (*box.Box property*), 8
left (*rect.Rect property*), 22
limit() (*box.Box method*), 10
location
    module, 14
Location (*class in location*), 14

LOWER_CENTER (*location.Location attribute*), 14
LOWER_LEFT (*location.Location attribute*), 14
LOWER_RIGHT (*location.Location attribute*), 15
lowerCenter() (*box.Box method*), 10
lowerLeft() (*box.Box method*), 10
lowerRight() (*box.Box method*), 10

M

mag (*transform.Transform property*), 21
MATH (*font.Font attribute*), 7
merge() (*box.Box method*), 10
mergePoint() (*box.Box method*), 10
MIL_SPEC (*font.Font attribute*), 7
mirrorX() (*box.Box method*), 10
mirrorX() (*location.Location method*), 15
mirrorY() (*box.Box method*), 10
mirrorY() (*location.Location method*), 15
module
    box, 8
    constants, 18
    dlo, 12
    dlogen, 20
    font, 7
    geo, 22
    grouping, 11
    instance, 25
    layer, 27
    location, 14
    namemapper, 8
    numeric, 24
    orientation, 18
    paramarray, 13
    pathstyle, 19
    physicalComponent, 28
    point, 16
    pointlist, 19
    polygon, 23
    rect, 21
    shape, 12
    signaltype, 26
    tech, 17
    termtype, 15
    text, 26
    transform, 20
    ulist, 14
moveBy() (*box.Box method*), 10
moveBy() (*grouping.Grouping method*), 12
moveBy() (*physicalComponent.PhysicalComponent method*), 28
moveBy() (*polygon.Polygon method*), 23
moveBy() (*rect.Rect method*), 22
moveBy() (*text.Text method*), 27
moveTo() (*box.Box method*), 10
moveTowards() (*box.Box method*), 10
MX (*orientation.Orientation attribute*), 18
MXR90 (*orientation.Orientation attribute*), 18
MY (*orientation.Orientation attribute*), 18
MYR90 (*orientation.Orientation attribute*), 18

N

name (*layer.Layer property*), 27
namemapper
    module, 8
NameMapper (*class in namemapper*), 8
number (*layer.Layer property*), 27
numeric
    module, 24
Numeric (*class in numeric*), 24

O

orientation
    module, 18
Orientation (*class in orientation*), 18
orientation (*transform.Transform property*), 21
OUTPUT (*termtype.TermType attribute*), 15
overlaps() (*box.Box method*), 10

P

paramarray
    module, 13
ParamArray (*class in paramarray*), 13
params_as_hash() (*dlo.PCellWrapper method*), 13
pathstyle
    module, 19
PathStyle (*class in pathstyle*), 19
PCellWrapper (*class in dlo*), 13
physicalComponent
    module, 28
PhysicalComponent (*class in physicalComponent*), 28
place() (*box.Box method*), 10
place() (*point.Point method*), 16
point
    module, 16
Point (*class in point*), 16
pointlist
    module, 19
PointList (*class in pointlist*), 19
polygon
    module, 23
Polygon (*class in polygon*), 23
POWER (*signaltype.SignalType attribute*), 26
produce() (*dlo.PCellWrapper method*), 13
purposeName (*layer.Layer property*), 27
purposeNumber (*layer.Layer property*), 27
PyCellContext (*class in dlo*), 13

R

R0 (*orientation.Orientation attribute*), 18

R180 (*orientation.Orientation attribute*), 18
R270 (*orientation.Orientation attribute*), 18
R90 (*orientation.Orientation attribute*), 18
RangeConstraint (*class in dlo*), 13
rect
    module, 21
Rect (*class in rect*), 21
register() (*tech.Tech method*), 17
REJECT (*in module constants*), 18
removeRegion() (*box.Box method*), 10
RESET (*signaltype.SignalType attribute*), 26
right (*box.Box property*), 8
right (*rect.Rect property*), 22
ROMAN (*font.Font attribute*), 7
rotate180() (*box.Box method*), 10
rotate180() (*location.Location method*), 15
rotate270() (*box.Box method*), 10
rotate270() (*location.Location method*), 15
rotate90() (*box.Box method*), 10
rotate90() (*location.Location method*), 15
ROUND (*pathstyle.PathStyle attribute*), 19

S

scale_factors (*numeric.Numeric property*), 25
scaleFactor (*numeric.Numeric property*), 24
scaleFormat() (*numeric.Numeric method*), 25
SCAN (*signaltype.SignalType attribute*), 26
SCRIPT (*font.Font attribute*), 7
set() (*box.Box method*), 10
set() (*point.Point method*), 16
set_shape() (*shape.Shape method*), 12
setAlignment() (*text.Text method*), 27
setBottom() (*box.Box method*), 10
setCenter() (*box.Box method*), 10
setCenterY() (*box.Box method*), 10
setCoord() (*box.Box method*), 10
setCoord() (*point.Point method*), 16
setDimension() (*box.Box method*), 10
setDrafting() (*text.Text method*), 27
setHeight() (*box.Box method*), 10
setLibName() (*dlogen.DloGen class method*), 20
setLocationPoint() (*box.Box method*), 11
setOrientation() (*instance.Instance method*), 25
setOrientation() (*text.Text method*), 27
setOrigin() (*instance.Instance method*), 26
setParams() (*instance.Instance method*), 25
setRange() (*box.Box method*), 11
setRangeX() (*box.Box method*), 11
setRangeY() (*box.Box method*), 11
setRect() (*box.Box method*), 11
setRight() (*box.Box method*), 11
setTech() (*dlogen.DloGen method*), 20
setTop() (*box.Box method*), 11
setWidth() (*box.Box method*), 11
setX() (*point.Point method*), 16
setY() (*point.Point method*), 17
shape
    module, 12
Shape (*class in shape*), 12
SIGNAL (*signaltype.SignalType attribute*), 26
signaltype
    module, 26
SignalType (*class in signaltype*), 26
snap() (*box.Box method*), 11
snap() (*point.Point method*), 17
snapTowards() (*box.Box method*), 11
snapTowards() (*point.Point method*), 17
snapX() (*box.Box method*), 11
snapX() (*point.Point method*), 17
snapY() (*box.Box method*), 11
snapY() (*point.Point method*), 17
STICK (*font.Font attribute*), 7
SWEDISH (*font.Font attribute*), 7
SWITCH (*termtype.TermType attribute*), 15

T

T (*in module ulist*), 14
tech
    module, 17
Tech (*class in tech*), 17
tech (*dlo.PyCellContext property*), 13
tech (*layer.Layer attribute*), 27
TechImpl (*class in tech*), 17
techInUse (*tech.Tech attribute*), 17
techsByName (*tech.Tech attribute*), 17
termtype
    module, 15
TermType (*class in termtype*), 15
text
    module, 26
Text (*class in text*), 26
TIEHI (*signaltype.SignalType attribute*), 26
TIELO (*signaltype.SignalType attribute*), 26
TIEOFF (*signaltype.SignalType attribute*), 26
toDiagAxes() (*point.Point method*), 17
toOrthogAxes() (*point.Point method*), 17
top (*box.Box property*), 8
top (*rect.Rect property*), 22
toString() (*grouping.Grouping method*), 12
toString() (*polygon.Polygon method*), 23
toString() (*rect.Rect method*), 22
transform
    module, 20
Transform (*class in transform*), 20
transform (*transform.Transform property*), 21
transform() (*box.Box method*), 11
transform() (*grouping.Grouping method*), 12
transform() (*location.Location method*), 15

transform() (*physicalComponent.PhysicalComponent method*), 29
transform() (*point.Point method*), 17
transform() (*polygon.Polygon method*), 23
transform() (*rect.Rect method*), 22
transform() (*text.Text method*), 27
TRISTATE (*termtype.TermType attribute*), 15
TRUNCATE (*pathstyle.PathStyle attribute*), 19

U

ulist
    module, 14
ulist (*class in ulist*), 14
UNUSED (*termtype.TermType attribute*), 15
UPPER_CENTER (*location.Location attribute*), 15
UPPER_LEFT (*location.Location attribute*), 14
UPPER_RIGHT (*location.Location attribute*), 15
upperCenter() (*box.Box method*), 11
upperLeft() (*box.Box method*), 11
upperRight() (*box.Box method*), 11
USE_DEFAULT (*in module constants*), 18

V

VARIABLE (*pathstyle.PathStyle attribute*), 19

X

x (*point.Point property*), 16
xOffset (*transform.Transform property*), 21

Y

y (*point.Point property*), 16
yOffset (*transform.Transform property*), 21