

---

# **BPG Documentation**

***Release 0.0.1***

**Pavan Bhargava**

**Sep 19, 2018**



**CONTENTS**

<b>1</b>	<b>BPG</b>	<b>1</b>
1.1	BPG package . . . . .	1
	<b>Python Module Index</b>	<b>31</b>
	<b>Index</b>	<b>33</b>



## 1.1 BPG package

### 1.1.1 Submodules

### 1.1.2 BPG.dataprep\_gdspy module

`BPG.dataprep_gdspy.cleanup_loop` (*coords\_list\_ori*, *eps\_grid*=0.0001)

**Parameters**

- **coords\_list\_ori** (*List[Tuple[float, float]]*) – The list of x-y coordinates composing a polygon shape
- **eps\_grid** – grid resolution below which points are considered to be the same

**Returns** **output\_dict** – Dictionary of ‘coords\_list\_out’ and ‘fully\_cleaned’

**Return type** Dict[str]

`BPG.dataprep_gdspy.coord_to_shapely` (*pos\_neg\_list\_list*)

Converts list of coordinate lists into shapely polygon objects

**Parameters** **pos\_neg\_list\_list** – The tuple of positive and negative lists of coordinate lists

**Returns** **polygon\_out** – The Shapely representation of the polygon

**Return type** Union[Polygon, Multipolygon]

`BPG.dataprep_gdspy.coords_apprx_in_line` (*coord1*, *coord2*, *coord3*, *eps\_grid*=0.0001)

Determines if three coordinates are in the same line

**Parameters**

- **coord1** (*Tuple[float, float]*) – First coordinate
- **coord2** (*Tuple[float, float]*) – Second coordinate
- **coord3** (*Tuple[float, float]*) – Third coordinate
- **eps\_grid** (*float*) – grid resolution below which points are considered to be the same

**Returns** True if coordinates are in a line, False if not in a line

**Return type** bool

`BPG.dataprep_gdspy.coords_cleanup` (*coords\_list\_ori*, *eps\_grid*=0.0001, *debug*=False)

clean up coordinates in the list that are redundant or harmful for following Shapely functions

**Parameters**

- **coords\_list\_ori** (*List[Tuple[float, float]]*) – list of coordinates that enclose a polygon
- **eps\_grid** (*float*) – a size smaller than the resolution grid size, if the difference of x/y coordinates of two points is smaller than it, these two points should actually share the same x/y coordinate
- **debug** (*bool*) –

**Returns** **coords\_list\_out** – The cleaned coordinate list

**Return type** List[Tuple[float, float]]

`BPG.dataprep_gdspy.dataprep_cleanup_gdspy (polygon, do_cleanup=True)`

Clean up a gdspy Polygon/PolygonSet by performing offset with size = 0

First offsets by size 0 with precision higher than the global grid size. Then calls an explicit rounding function to the grid size. This is done because it is unclear how the clipper/gdspy library handles precision

#### Parameters

- **polygon** (*Union[gdspy.Polygon, gdspy.PolygonSet]*) – The polygon to clean
- **do\_cleanup** (*bool*) – True to perform the cleanup. False will return input polygon unchanged

**Returns** **clean\_polygon** – The cleaned up polygon

**Return type** Union[gdspy.Polygon, gdspy.PolygonSet]

`BPG.dataprep_gdspy.dataprep_coord_to_gdspy (pos_neg_list_list, manh_grid_size, do_manh)`

Converts list of polygon coordinate lists into GDSPY polygon objects The expected input list will be a list of all polygons on a given layer

#### Parameters

- **pos\_neg\_list\_list** (*Tuple[List, List]*) – A tuple containing two lists: the list of positive polygon shapes and the list of negative polygon shapes. Each polygon shape is a list of point tuples
- **manh\_grid\_size** (*float*) – The Manhattanization grid size
- **do\_manh** (*bool*) – True to perform Manhattanization

**Returns** **polygon\_out** – The gdspy.Polygon formatted polygons

**Return type** Union[gdspy.Polygon, gdspy.PolygonSet]

`BPG.dataprep_gdspy.dataprep_oversize_gdspy (polygon, offset)`

Grow a polygon by an offset. Perform cleanup to ensure proper polygon shape.

#### Parameters

- **polygon** (*Union[gdspy.Polygon, gdspy.PolygonSet, None]*) – The polygon to size, in gdspy representation
- **offset** (*float*) – The amount to grow the polygon

**Returns** **polygon\_oversized** – The oversized polygon

**Return type** Union[gdspy.Polygon, gdspy.PolygonSet, None]

`BPG.dataprep_gdspy.dataprep_roughsize_gdspy (polygon, size_amount, do_manh)`

Add a new polygon that is rough sized by 'size\_amount' from the provided polygon. Rough sizing entails:

- oversize by 2x the global rough grid size

- undersize by 2x the global rough grid size
- oversize by the global rough grid size
- Manhattanize to the global rough grid
- undersize by the fine global fine grid size
- oversize by the fine global fine grid size
- oversize by 'size\_amount' less the 2x global grid size already used

#### Parameters

- **polygon** (*Union[gdspy.Polygon, gdspy.PolygonSet]*) – polygon to be used as the base shape for the rough add, in gdspy representation
- **size\_amount** (*float*) – amount to oversize (undersize is not supported, will be set to 0 if negative) the rough added shape
- **do\_manh** (*bool*) – True to perform Manhattanization of after the oouuo shape

**Returns** **polygon\_roughsized** – the rough added polygon shapes, in gdspy representation

**Return type** Union[gdspy.Polygon, gdspy.PolygonSet]

BPG.dataprep\_gdspy.**dataprep\_undersize\_gdspy** (*polygon, offset*)

Shrink a polygon by an offset. Perform cleanup to ensure proper polygon shape.

#### Parameters

- **polygon** (*Union[gdspy.Polygon, gdspy.PolygonSet, None]*) – The polygon to size, in gdspy representation
- **offset** (*float*) – The amount to shrink the polygon

**Returns** **polygon\_undersized** – The undersized polygon

**Return type** Union[gdspy.Polygon, gdspy.PolygonSet, None]

BPG.dataprep\_gdspy.**gdspy\_manh** (*polygon\_gdspy, manh\_grid\_size, do\_manh*)

Performs Manhattanization on a gdspy representation of a polygon, and returns a gdspy representation of the Manhattanized polygon

#### Parameters

- **polygon\_gdspy** (*Union[gdspy.Polygon, gdspy.PolygonSet, None]*) – The gdspy representation of the polygons to be Manhattanized
- **manh\_grid\_size** (*float*) – grid size for Manhattanization, edge length after Manhattanization should be larger than it
- **do\_manh** (*bool*) – True to perform Manhattanization

**Returns** **polygon\_out** – The Manhattanized polygon, in gdspy representation

**Return type** Union[gdspy.Polygon, gdspy.PolygonSet]

BPG.dataprep\_gdspy.**manh\_skill** (*poly\_coords, manh\_grid\_size, manh\_type*)

Convert a polygon into a polygon with orthogonal edges (ie, performs Manhattanization)

#### Parameters

- **poly\_coords** (*List[Tuple[float, float]]*) – list of coordinates that enclose a polygon

- **manh\_grid\_size** (*float*) – grid size for Manhattanization, edge length after Manhattanization should be larger than it
- **manh\_type** (*str*) – ‘inc’ : the Manhattanized polygon is larger compared to the one on the manh grid ‘dec’ : the Manhattanized polygon is smaller compared to the one on the manh grid ‘non’ : additional feature, only map the coords to the manh grid but do no Manhattanization

**Returns** **poly\_coords\_cleanup** – The Manhattanized list of coordinates describing the polygon

**Return type** List[Tuple[float, float]]

`BPG.dataprep_gdspy.not_manh(coord_list, eps_grid=1e-06, print_failing_points=False)`

Checks whether the passed coordinate list is Manhattanized

**Parameters**

- **coord\_list** (*List[Tuple[float, float]]*) – The coordinate list to check
- **eps\_grid** (*float*) – The grid tolerance below which points are considered the same
- **print\_failing\_points** (*bool*) – True to print the coordinates of the points that do not have Manhattanized edges

**Returns** **non\_manh\_edge** – The count of number of edges that are non-Manhattan in this shape

**Return type** int

`BPG.dataprep_gdspy.poly_operation(polygon1, polygon2, operation, size_amount, do_manh=False)`

**Parameters**

- **polygon1** (*Union[gdspy.Polygon, gdspy.PolygonSet, None]*) – The shapes currently on the output layer
- **polygon2** (*Union[gdspy.Polygon, gdspy.PolygonSet, None]*) – The shapes on the input layer that will be added/subtracted to/from the output layer
- **operation** (*str*) – The operation to perform: ‘rad’, ‘add’, ‘sub’, ‘ext’, ‘ouo’, ‘del’
- **size\_amount** (*float*) – The amount to over/undersize the shapes to be added/subtracted
- **do\_manh** (*bool*) – True to perform Manhattanization during the ‘rad’ operation

**Returns** **polygons\_out** – The new polygons present on the output layer

**Return type** Union[gdspy.Polygon, gdspy.PolygonSet, None]

`BPG.dataprep_gdspy.polyop_gdspy_to_point_list(polygon_gdspy_in, fracture=True, do_manh=True, manh_grid_size=0.001, debug=False)`

Converts the gdspy representation of the polygon into a list of fractured polygon point lists

**Parameters**

- **polygon\_gdspy\_in** (*Union[gdspy.Polygon, gdspy.PolygonSet]*) – The gdspy polygons to be converted to lists of coordinates
- **fracture** (*bool*) – True to fracture shapes
- **do\_manh** (*bool*) – True to perform Manhattanization
- **manh\_grid\_size** (*float*) – The Manhattanization grid size
- **debug** (*bool*) – True to print debug information



**Returns output\_list\_of\_coord\_lists** – A list containing the polygon point lists that compose the input gdspy polygon

**Return type** List[List[Tuple[float, float]]]

`BPG.dataprep_gdspy.shapely_to_gdspy(geom_shapely)`

Convert the shapely representation of a polygon/multipolygon into the gdspy representation of the polygon/polygonset

**Parameters** `geom_shapely` (`Union[Polygon, MultiPolygon]`) – The shapely representation of the polygon

**Returns** `polygon_gdspy` – The gdspy representation of the polygon

**Return type** Union[gdspy.Polygon, gdspy.PolygonSet]

`BPG.dataprep_gdspy.shapely_to_gdspy_polygon(polygon_shapely)`

Converts the shapely representation of a polygon to a gdspy representation

**Parameters** `polygon_shapely` (`shapely.geometry.Polygon`) – The shapely representation of the polygon

**Returns** `polygon_gdspy` – The gdspy representation of the polygon

**Return type** gdspy.Polygon

`BPG.dataprep_gdspy.simplify_coord_to_gdspy(pos_neg_list_list, tolerance=0.0005)`

Simplifies a polygon coordinate-list representation of a complex polygon (multiple shapes, with holes, etc) and converts the simplified polygon into gdspy representation. Simplification involves reducing the number of points in the shape based on a tolerance of how far the points are from being collinear.

**Parameters**

- **pos\_neg\_list\_list** (`Tuple[List[List[Tuple[float, float]]], List[List[Tuple[float, float]]]]`) – Tuple containing the positive and negative list of polygon point-lists
- **tolerance** (`float`) – The tolerance within which a set of points are deemed collinear

**Returns** `poly_gdspy_simplified` – The simplified polygon in gdspy representation

**Return type** Union[gdspy.PolygonSet, gdspy.Polygon]

### 1.1.3 BPG.dataprep\_shapely module

### 1.1.4 BPG.lumerical\_generator module

Module containing various classes used to systematically generate clean Lumerical script code

**class** `BPG.lumerical_generator.LumericalCodeGenerator`

Bases: object

This class enables the generation of lumerical .lsf code

**add\_code** (`code: str`) → None

Adds provided statement of code to the script file, and formats it accordingly Adds a semicolon and a newline character to each line to match standard LSF syntax

**Parameters** `code` (`str`) – Single string containing lumerical script

**add\_code\_block** (`code: List[str]`)

Adds a preformatted list of lines of code to the script file

**Parameters** `code` (*List[str]*) – Single string containing numerical script

**add\_line** (*code: str*) → None

Adds provided line of code to the script file Does not add a semicolon, but does add a newline character

**Parameters** `code` (*str*) – Single string containing numerical script

**get\_file\_header** ()

Returns a list of strings that form the header of the script file

**Returns** `header` – Contains comments for the header of the file

**Return type** *List[str]*

**set** (*key: str, value*) → None

Conveniently adds a set statement to the LSF file

**Parameters**

- **key** (*str*) – parameter to be changed with the set statement
- **value** (*any*) – value that the parameter will be assigned

**class** `BPG.lumerical_generator.LumericalDesignGenerator` (*filepath*)

Bases: `BPG.lumerical_generator.LumericalCodeGenerator`

**export\_to\_lsf** ()

Take all code in the database and export it to a numerical script file

**class** `BPG.lumerical_generator.LumericalSweepGenerator` (*filepath*)

Bases: `BPG.lumerical_generator.LumericalCodeGenerator`

This class enables the creation of .lsf files for swept variables

**add\_sweep\_point** (*script\_name*)

Adds a given script name to the be run in the main sweep loop. Scripts are executed in the order in which they are added

**Parameters** `script_name` (*str*) – Name of script to be executed

**create\_sweep\_loop** ()

**export\_to\_lsf** ()

Take all code in the database and export it to a numerical script file

### 1.1.5 BPG.lumerical\_sim module

**class** `BPG.lumerical_sim.FDESolver`

Bases: `BPG.lumerical_sim.LumericalSimObj`

Lumerical Simulation Object for Finite-Difference Eigenmode Solver

**align\_to\_port** (*port, offset=(0, 0), align\_orient=True*)

Moves the center of the simulation object to align to the provided photonic port. Overrides the superclass method to support setting the orientation to match port

**Parameters**

- **port** (`PhotonicPort`) – Photonic port for the simulation object to be aligned to
- **offset** (*Tuple*) – (x, y) offset relative to the port location
- **align\_orient** (*bool*) – True to set the orientation to match the port orientation, False to ignore port orientation

**lsf\_export()**

Returns a list of Lumerical code describing the creation of a FDESolver object

**Returns** **lsf\_code** – list of Lumerical code to create the FDESolver object

**Return type** List[str]

**mesh\_size**

**orientation**

**class** BPG.lumerical\_sim.FDTDSolver

Bases: *BPG.lumerical\_sim.LumericalSimObj*

Lumerical Simulation Object for Finite-Difference Time Domain Solver

**lsf\_export()**

Returns a list of Lumerical code describing the creation of a FDESolver object

**Returns** **lsf\_code** – list of Lumerical code to create the FDESolver object

**Return type** List[str]

**class** BPG.lumerical\_sim.LumericalSimObj

Bases: *BPG.photonic\_core.Box*, *BPG.lumerical\_generator.LumericalCodeGenerator*

Abstract Base Class for all simulation/monitor objects in Lumerical

All simulation objects have a common representation for geometry and generate lsf code, so LumericalSimObj inherits both from Box and from LumericalCodeGenerator

**content**

Return self so that lsf\_export can be called by BAG from the content list

**lsf\_export()** → List[str]

Returns a list of Lumerical code describing the creation of the simulation object

Unlike the export\_lsf method for photonic objects, this method is not a classmethod, and relies on internal access to the instances attributes

### 1.1.6 BPG.lumerical\_tb module

**class** BPG.lumerical\_tb.LumericalTB(*temp\_db, lib\_name, params, used\_names, \*\*kwargs*)

Bases: *BPG.photonic\_template.PhotonicTemplateBase*

**add\_EME\_port()**

**add\_EME\_solver()**

**add\_FDE\_solver()** → BPG.lumerical\_sim.FDESolver

Create a blank FDE solver, add it to the db, and return it to the user for manipulation

**add\_FDTD\_port()**

**add\_FDTD\_solver()**

Create a blank FDTD solver, add it to the db, and return it to the user for manipulation

**add\_effective\_index\_monitor()**

**add\_eme\_profile()**

**add\_freq\_domain\_monitor()**

**add\_gaussian\_source()**

**add\_index\_monitor()**

`add_mode_expansion_monitor()`  
`add_mode_source()`  
`add_movie_monitor()`  
`add_point_source()`  
`add_time_domain_monitor()`  
`add_total_field_source()`  
`add_var_FDTD_solver()`  
`construct_tb()`  
Override this method to specify the procedure for generating the testbench and simulation. YOU MUST IMPLEMENT THIS METHOD TO CREATE A TESTBENCH!  
`create_dut()`  
Create and place the provided layout class and parameters at the origin  
`draw_layout()`  
This method is used internally to assemble the instance and the TB sources. DO NOT CALL THIS  
`classmethod get_params_info()`  
Returns a dictionary from parameter names to descriptions.  
**Returns** `param_info` – dictionary from parameter names to descriptions.  
**Return type** Optional[Dict[str, str]]  
`plane_wave_source()`

### 1.1.7 BPG.manh\_shapely module

`BPG.manh_shapely.cleanup_loop(coords_list_ori, eps_grid=0.0001)`

`BPG.manh_shapely.coords_apprx_in_line(coord1, coord2, coord3, eps_grid=0.0001)`

Tell if three coordinates are in the same line Expected to have three consecutive coordinates as inputs when the function is called

`BPG.manh_shapely.coords_cleanup(coords_list_ori, eps_grid=0.0001, debug=False)`

clean up coordinates in the list that are redundant or harmful for following Shapely functions

#### Parameters

- **coords\_list\_ori** (`list[tuple[float, float]]`) – list of coordinates that enclose a polygon
- **eps\_grid** (`float`) – a size smaller than the resolution grid size, if the difference of x/y coordinates of two points is smaller than it, these two points should actually share the same x/y coordinate
- **debug** (`bool`) –

`BPG.manh_shapely.manh_skill(poly_coords, manh_grid_size, manh_type)`

Convert a polygon into the polygon with orthogonal edges, detailed flavors are the same as it is in the SKILL code

#### Parameters

- **poly\_coords** (`list[tuple[float, float]]`) – list of coordinates that enclose a polygon

- **manh\_grid\_size** (*float*) – grid size for manhattanization, edge length after manhattanization should be larger than it
- **manh\_type** (*str*) – ‘inc’ : the manhattanized polygon is larger compared to the one on the manh grid ‘dec’ : the manhattanized polygon is smaller compared to the one on the manh grid ‘non’ : additional feature, only map the coords to the manh grid but do no manhattanization

`BPG.manh_shapely.plot_coords(ax, x, y, color='#999999', zorder=1)`

`BPG.manh_shapely.plot_line(ax, ob, color='r')`

`BPG.manh_shapely.polyop_manh(geom, manh_grid_size, do_manh)`

`BPG.manh_shapely.polyop_manh_polygon(geom, manh_grid_size, do_manh)`

### 1.1.8 BPG.photonic\_core module

**class** `BPG.photonic_core.Box`

Bases: `object`

A class representing a 3D rectangle

**align\_to\_port** (*port*, *offset*=(0, 0))

Moves the center of the simulation object to align to the provided photonic port

#### Parameters

- **port** (`PhotonicPort`) – Photonic port for the simulation object to be aligned to
- **offset** (*Tuple*) – (x, y) offset relative to the port location

**move\_by** (*dx*, *dy*, *unit\_mode*=False)

**set\_center\_span** (*dim*, *center*, *span*)

Sets the center and span of a given geometry dimension

#### Parameters

- **dim** (*str*) – ‘x’, ‘y’, or ‘z’ for the corresponding dimension
- **center** (*float*) – coordinate location of the center of the geometry
- **span** (*float*) – length of the geometry along the dimension

**set\_span** (*dim*, *span*)

Sets the span of a given geometry dimension

#### Parameters

- **dim** (*str*) – ‘x’, ‘y’, or ‘z’ for the corresponding dimension
- **span** (*float*) – length of the geometry along the dimension

**class** `BPG.photonic_core.CoordBase`

Bases: `object`

A class representing the basic unit of measurement for all objects in BPG.

All user-facing values are assumed to be floating point numbers in units of microns. BAG internal functions assume that we receive ‘unit-mode’ numbers, which are integers in units of nanometers. Both formats are supported.

**float**

Returns the rounded floating point number closest to a valid point on the resolution grid

**meters**

Returns the rounded floating point number in meters closest to a valid point on the resolution grid

**micron** = `Decimal('0.000001')`

**microns**

**res** = `Decimal('0.001')`

**unit\_mode**

**value**

**class** `BPG.photonic_core.PTech`

Bases: `object`

**finalize\_template** (*a*)

**get\_layer\_id** (*layer*)

**use\_flip\_parity** ()

**class** `BPG.photonic_core.PhotonicBagLayout` (*grid, use\_cybagoa=False*)

Bases: `bag.layout.core.BagLayout`

This class contains layout information of a cell.

#### Parameters

- **grid** (`bag.layout.routing.RoutingGrid`) – the routing grid instance.
- **use\_cybagoa** (*bool*) – True to use cybagoa package to accelerate layout.

**add\_monitor\_obj** (*monitor\_obj*)

Add a new Lumerical monitor object to the db

**add\_round** (*round\_obj*)

Add a new (arrayed) round shape.

**Parameters** **round\_obj** (`BPG.photonic_objects.PhotonicRound`) – the round object to add.

**add\_sim\_obj** (*sim\_obj*)

Add a new Lumerical simulation object to the db

**add\_source\_obj** (*source\_obj*)

Add a new Lumerical source object to the db

**finalize** ()

Prevents any further changes to this layout.

**get\_content** (*lib\_name, cell\_name, rename\_fun*)

returns a list describing geometries in this layout.

#### Parameters

- **lib\_name** (*str*) – the layout library name.
- **cell\_name** (*str*) – the layout top level cell name.
- **rename\_fun** (*Callable[[str], str]*) – the layout cell renaming function.

**Returns** **content** – a list describing this layout, or `PyOALayout` if cybagoa package is enabled.

**Return type** `Union[List[Any], Tuple[str, 'cybagoa.PyOALayout']]`

**move\_all\_by** (*dx=0.0, dy=0.0, unit\_mode=False*)

Move all layout objects in this layout by the given amount.

**Parameters**

- **dx** (*Union[float, int]*) – the X shift.
- **dy** (*Union[float, int]*) – the Y shift.
- **unit\_mode** (*bool*) – True if shift values are given in resolution units.

**class** BPG.photonic\_core.**PhotonicBagProject** (*bag\_config\_path=None, port=None*)

Bases: bag.core.BagProject

The main bag controller class.

This class extracts user configuration variables and issues high level bag commands. Most config variables have defaults pointing to files in the BPG/examples/tech folder

**Parameters**

- **bag\_config\_path** (*Optional[str]*) – the bag configuration file path. If None, will attempt to read from environment variable BAG\_CONFIG\_PATH.
- **port** (*Optional[int]*) – the BAG server process port number. If not given, will read from port file.

**static load\_yaml** (*filepath*)

Setup standardized method for yaml loading

**class** BPG.photonic\_core.**Plane**

Bases: object

A class representing a plane that is orthogonal to one of the cardinal axes

TODO: Implement this class

**class** BPG.photonic\_core.**XY**

Bases: object

A class representing a single point on the XY plane

**x**

**x\_float**

**x\_meters**

**xy**

**xy\_float**

**xy\_meters**

**y**

**y\_float**

**y\_meters**

**class** BPG.photonic\_core.**XYZ**

Bases: object

A class representing a single point on the XYZ space

**x**

**x\_float**

**x\_meters**

**xyz**

```
xyz_float
xyz_meters
y
y_float
y_meters
z
z_float
z_meters
```

### 1.1.9 BPG.photonic\_layout\_manager module

**class** BPG.photonic\_layout\_manager.**PhotonicLayoutManager** (*bprj, spec\_file*)  
Bases: bag.simulation.core.DesignManager

Class that manages the creation of Photonic Layouts and Lumerical LSF files

**dataprep** (*debug=False*)

**Parameters** **debug** (*bool*) – True to print debug information

**generate\_flat\_gds** (*generate\_gds=True, layout\_params\_list=None, cell\_name\_list=None, gen\_full\_gds=False, gen\_design\_gds=True, gen\_physical\_gds=True, debug=False*) → None

Generates a batch of layouts with the layout package/class in the spec file with the parameters set by layout\_params\_list and names them according to cell\_name\_list. Each dict in the layout\_params\_list creates a new layout

**Parameters**

- **generate\_gds** (*Optional[bool]*) – Optional parameter: True (default) to generate the GDS
- **layout\_params\_list** (*List[dict]*) – Optional list of dicts corresponding to layout parameters passed to the generator class
- **cell\_name\_list** (*List[str]*) – Optional list of strings corresponding to the names given to each generated layout
- **gen\_full\_gds** (*bool*) – True to generate a gds with both physical and design layers
- **gen\_design\_gds** (*bool*) – True to generate the gds with only photonic design (and port) layers
- **gen\_physical\_gds** (*bool*) – True to generate the gds with only physical layers
- **debug** (*bool*) – True to print debug information

**generate\_gds** (*layout\_params\_list=None, cell\_name\_list=None*) → None

Generates a batch of layouts with the layout package/class in the spec file with the parameters set by layout\_params\_list and names them according to cell\_name\_list. Each dict in the layout\_params\_list creates a new layout

**Parameters**

- **layout\_params\_list** (*List[dict]*) – Optional list of dicts corresponding to layout parameters passed to the generator class



- **cell\_name\_list** (*List[str]*) – Optional list of strings corresponding to the names given to each generated layout

**generate\_lsf** (*debug=False*)

Converts generated layout to lsf format for lumerical import

**generate\_tb** (*generate\_gds=False, debug=False*)

Generates the lumerical testbench lsf

**static load\_yaml** (*filepath*)

Setup standardized method for yaml loading

**make\_tdb** () → None

Makes a new PhotonicTemplateDB instance assuming all contained layouts are generated independently of the grid

### 1.1.10 BPG.photonic\_objects module

This module defines various layout objects one can add and manipulate in a template.

**class** BPG.photonic\_objects.**PhotonicAdvancedPolygon** (*resolution, layer, points, negative\_points, unit\_mode=False*)

Bases: bag.layout.objects.Polygon

A layout polygon object.

#### Parameters

- **resolution** (*float*) – the layout grid resolution.
- **layer** (*Union[str, Tuple[str, str]]*) – the layer name, or a tuple of layer name and purpose name. If purpose name not given, defaults to ‘drawing’.
- **points** (*List[Tuple[Union[float, int], Union[float, int]]]*) – the points defining the polygon.
- **unit\_mode** (*bool*) – True if the points are given in resolution units.

**class** BPG.photonic\_objects.**PhotonicBlockage** (*resolution, block\_type, block\_layer, points, unit\_mode=False*)

Bases: bag.layout.objects.Blockage

A blockage object.

Subclass Polygon for code reuse.

#### Parameters

- **resolution** (*float*) – the layout grid resolution.
- **block\_type** (*str*) – the blockage type. Currently supports ‘routing’ and ‘placement’.
- **block\_layer** (*str*) – the blockage layer. This value is ignored if blockage type is ‘placement’.
- **points** (*List[Tuple[Union[float, int], Union[float, int]]]*) – the points defining the blockage.
- **unit\_mode** (*bool*) – True if the points are given in resolution units.

**classmethod from\_content** (*content, resolution*)

```
class BPG.photonic_objects.PhotonicBoundary (resolution, boundary_type, points,  
                                              unit_mode=False)
```

Bases: bag.layout.objects.Boundary

A boundary object.

Subclass Polygon for code reuse.

#### Parameters

- **resolution** (*float*) – the layout grid resolution.
- **boundary\_type** (*str*) – the boundary type. Currently supports ‘PR’, ‘snap’, and ‘area’.
- **points** (*List[Tuple[Union[float, int], Union[float, int]]]*) – the points defining the blockage.
- **unit\_mode** (*bool*) – True if the points are given in resolution units.

```
classmethod from_content (content, resolution)
```

```
class BPG.photonic_objects.PhotonicInstance (parent_grid, lib_name, master, loc, orient,  
                                              name=None, nx=1, ny=1, spx=0, spy=0,  
                                              unit_mode=False)
```

Bases: bag.layout.objects.Instance

A photonic layout instance, with optional arraying parameters.

#### Parameters

- **parent\_grid** (*RoutingGrid*) – the parent RoutingGrid object.
- **lib\_name** (*str*) – the layout library name.
- **master** (*TemplateBase*) – the master template of this instance.
- **loc** (*Tuple[Union[float, int], Union[float, int]]*) – the origin of this instance.
- **orient** (*str*) – the orientation of this instance.
- **name** (*Optional[str]*) – name of this instance.
- **nx** (*int*) – number of columns.
- **ny** (*int*) – number of rows.
- **spx** (*Union[float, int]*) – column pitch.
- **spy** (*Union[float, int]*) – row pitch.
- **unit\_mode** (*bool*) – True if layout dimensions are specified in resolution units.

#### content

A dictionary representation of this instance.

```
get_bound_box_of (row=0, col=0)
```

Returns the bounding box of an instance in this mosaic.

```
get_photonic_port (name, row=0, col=0)
```

Returns the photonic port object associated with the provided port name

#### Parameters

- **name** (*str*) – name of the port to be returned
- **row** (*int*) – row in the array of instances to be accessed
- **col** (*int*) – column in the array of instances to be accessed

**Returns** `port` – photonic port object associated with the provided name

**Return type** *PhotonicPort*

**get\_port\_used** (*port\_name*)

**master**

The master template of this instance.

**move\_by** (*dx=0, dy=0, unit\_mode=False*)

Move this instance by the given amount.

**Parameters**

- **dx** (*Union[float, int]*) – the X shift.
- **dy** (*Union[float, int]*) – the Y shift.
- **unit\_mode** (*bool*) – True if shifts are given in resolution units

**set\_port\_used** (*port\_name*)

**transform** (*loc=(0, 0), orient='R0', unit\_mode=False, copy=False*)

Transform this figure.

**class** BPG.photonic\_objects.**PhotonicInstanceInfo** (*res, change\_orient=True, \*\*kwargs*)

Bases: bag.layout.objects.InstanceInfo

A dictionary that represents a layout instance.

**copy** ()

Override copy method of InstanceInfo to return a PhotonicInstanceInfo instead.

**master\_key**

**class** BPG.photonic\_objects.**PhotonicPath** (*resolution, layer, width, points, end\_style='truncate', join\_style='extend', unit\_mode=False*)

Bases: bag.layout.objects.Figure

A layout path. Only 45/90 degree turns are allowed.

**Parameters**

- **resolution** (*float*) – the layout grid resolution.
- **layer** (*string or (string, string)*) – the layer name, or a tuple of layer name and purpose name. If purpose name not given, defaults to ‘drawing’.
- **width** (*float*) – width of this path, in layout units.
- **points** (*List[Tuple[float, float]]*) – list of path points.
- **end\_style** (*str*) – the path ends style. Currently support ‘truncate’, ‘extend’, and ‘round’.
- **join\_style** (*str*) – the ends style at intermediate points of the path. Currently support ‘extend’ and ‘round’.
- **unit\_mode** (*bool*) – True if width and points are given as resolution units instead of layout units.

**content**

A dictionary representation of this path.

**classmethod** **from\_content** (*content, resolution*)

**layer**

The rectangle (layer, purpose) pair.

**lower**

**move\_by** (*dx=0, dy=0, unit\_mode=False*)

Move this path by the given amount.

**Parameters**

- **dx** (*float*) – the X shift.
- **dy** (*float*) – the Y shift.
- **unit\_mode** (*bool*) – True if shifts are given in resolution units.

**points**

**points\_unit**

**classmethod polygon\_pointlist\_export** (*vertices*)

**Parameters** **vertices** (*List[Tuple[float, float]]*) – The verticies from the content list of this polygon

**Returns** **output\_list** – The positive and negative polygon pointlists describing this polygon

**Return type** *Tuple[List, List]*

**polygon\_points**

**process\_points** (*pts, width, eps=1e-05, unit\_mode=False*)

**Parameters**

- **pts** –
- **width** –
- **eps** –
- **unit\_mode** –

**ttransform** (*loc=(0, 0), orient='R0', unit\_mode=False, copy=False*)

Transform this figure.

**upper**

**valid**

Returns True if this instance is valid.

**width**

**width\_unit**

**class** BPG.photonic\_objects.**PhotonicPathCollection** (*resolution, paths*)

Bases: *bag.layout.objects.PathCollection*

A layout figure that consists of one or more paths.

This class make it easy to draw bus/trasmission line objects.

**Parameters**

- **resolution** (*float*) – layout unit resolution.
- **paths** (*List[Path]*) – paths in this collection.

```
class BPG.photonic_objects.PhotonicPinInfo (res, **kwargs)
    Bases: bag.layout.objects.PinInfo

    A dictionary that represents a layout pin.

    classmethod from_content (content, resolution)

    param_list = ['net_name', 'pin_name', 'label', 'layer', 'bbox', 'make_rect']

    transform (loc, orient, unit_mode, copy)
```

```
class BPG.photonic_objects.PhotonicPolygon (resolution, layer, points, unit_mode=False)
    Bases: bag.layout.objects.Polygon

    A layout polygon object.
```

#### Parameters

- **resolution** (*float*) – the layout grid resolution.
- **layer** (*Union[str, Tuple[str, str]]*) – the layer name, or a tuple of layer name and purpose name. If purpose name not given, defaults to ‘drawing’.
- **points** (*List[Tuple[Union[float, int], Union[float, int]]]*) – the points defining the polygon.
- **unit\_mode** (*bool*) – True if the points are given in resolution units.

```
classmethod from_content (content, resolution)
```

```
classmethod lsf_export (vertices, layer_prop) → List[str]
```

Describes the current polygon shape in terms of lsf parameters for lumerical use

#### Parameters

- **vertices** (*List[Tuple[float, float]]*) – ordered list of x,y coordinates representing the points of the polygon
- **layer\_prop** (*dict*) – dictionary containing material properties for the desired layer

**Returns** **lsf\_code** – list of str containing the lsf code required to create specified rectangles

**Return type** List[str]

```
classmethod polygon_pointlist_export (vertices)
```

**Parameters** **vertices** (*List[Tuple[float, float]]*) – The vertices from the content list of this polygon

**Returns** **output\_list** – The positive and negative polygon pointlists describing this polygon

**Return type** Tuple[List, List]

```
class BPG.photonic_objects.PhotonicRect (layer, bbox, nx=1, ny=1, spx=0, spy=0, unit_mode=False)
```

Bases: bag.layout.objects.Rect

A layout rectangle, with optional arraying parameters.

#### Parameters

- **layer** (*string or (string, string)*) – the layer name, or a tuple of layer name and purpose name. If purpose name not given, defaults to ‘drawing’.
- **bbox** (*bag.layout.util.BBox or bag.layout.util.BBoxArray*) – the base bounding box. If this is a BBoxArray, the BBoxArray’s arraying parameters are used.
- **nx** (*int*) – number of columns.

- **ny** (*int*) – number of rows.
- **spx** (*float*) – column pitch.
- **spy** (*float*) – row pitch.
- **unit\_mode** (*bool*) – True if layout dimensions are specified in resolution units.

**classmethod from\_content** (*content, resolution*)

**classmethod lsf\_export** (*bbox, layer\_prop, nx=1, ny=1, spx=0.0, spy=0.0*) → List[str]

Describes the current rectangle shape in terms of lsf parameters for lumerical use. Note that Lumerical uses meters as the base unit, and all input coords are assumed to be in microns. This method inherently resizes

#### Parameters

- **bbox** ([[*float, float*], [*float, float*]]) – lower left and upper right corner xy coordinates
- **layer\_prop** (*dict*) – dictionary containing material properties for the desired layer
- **nx** (*int*) – number of arrayed rectangles in the x-direction
- **ny** (*int*) – number of arrayed rectangles in the y-direction
- **spx** (*float*) – space between arrayed rectangles in the x-direction
- **spy** (*float*) – space between arrayed rectangles in the y-direction

**Returns lsf\_code** – list of str containing the lsf code required to create specified rectangles

**Return type** List[str]

**classmethod polygon\_pointlist\_export** (*bbox, nx=1, ny=1, spx=0.0, spy=0.0*)

Convert the PhotonicRect geometry to a list of polygon pointlists.

#### Parameters

- **bbox** ([[*float, float*], [*float, float*]]) – lower left and upper right corner xy coordinates
- **nx** (*int*) – number of arrayed rectangles in the x-direction
- **ny** (*int*) – number of arrayed rectangles in the y-direction
- **spx** (*float*) – space between arrayed rectangles in the x-direction
- **spy** (*float*) – space between arrayed rectangles in the y-direction

**Returns output\_list** – The positive and negative polygon pointlists describing the photonicRect

**Return type** Tuple[List, List]

**class** BPG.photonic\_objects.**PhotonicRound** (*layer, resolution, center, rout, rin=0, theta0=0, theta1=360, nx=1, ny=1, spx=0, spy=0, unit\_mode=False*)

Bases: bag.layout.objects.Arrayable

A layout round object, with optional arraying parameters.

#### Parameters

- **layer** (*string or (string, string)*) – the layer name, or a tuple of layer name and purpose name. If purpose name not given, defaults to ‘drawing’.
- **rout** –
- **rin** –

- **theta0** –
- **theta1** –
- **nx** (*int*) – number of columns.
- **ny** (*int*) – number of rows.
- **spx** (*float*) – column pitch.
- **spy** (*float*) – row pitch.
- **unit\_mode** (*bool*) – True if layout dimensions are specified in resolution units.

**center**

The center in layout units

**center\_unit**

The center in resolution units

**content**

A dictionary representation of this rectangle.

**classmethod from\_content** (*content, resolution*)

**layer**

The rectangle (layer, purpose) pair.

**classmethod lsf\_export** (*rout, rin, theta0, theta1, layer\_prop, center, nx=1, ny=1, spx=0.0, spy=0.0*)

**Parameters**

- **rout** –
- **rin** –
- **theta0** –
- **theta1** –
- **layer\_prop** –
- **center** –
- **nx** –
- **ny** –
- **spx** –
- **spy** –

**move\_by** (*dx=0, dy=0, unit\_mode=False*)

Moves the round object

**static num\_of\_sparse\_point\_round** (*radius, res\_grid\_size*)

**classmethod polygon\_pointlist\_export** (*rout, rin, theta0, theta1, center, nx=1, ny=1, spx=0.0, spy=0.0, resolution=0.001*)

**rin**

The inner radius in layout units

**rin\_unit**

The inner radius in resolution units

**rout**

The outer radius in layout units

**rout\_unit**

The outer radius in resolution units

**theta0**

The starting angle, in degrees

**theta1**

The ending angle, in degrees

**transform** (*loc*=(0, 0), *orient*='R0', *unit\_mode*=False, *copy*=False)

Transform this figure.

**class** BPG.photonic\_objects.**PhotonicTLineBus** (*resolution, layer, points, widths, spaces, end\_style='truncate', unit\_mode=False*)

Bases: bag.layout.objects.TLineBus

A transmission line bus drawn using Path.

assumes only 45 degree turns are used, and begin and end line segments are straight.

#### Parameters

- **resolution** (*float*) – layout unit resolution.
- **layer** (*Union[str, Tuple[str, str]]*) – the bus layer.
- **points** (*List[Tuple[Union[float, int], Union[float, int]]]*) – list of center points of the bus.
- **widths** (*List[Union[float, int]]*) – list of wire widths. 0 index is left/bottom most wire.
- **spaces** (*List[Union[float, int]]*) – list of wire spacings.
- **end\_style** (*str*) – the path ends style. Currently support 'truncate', 'extend', and 'round'.
- **unit\_mode** (*bool*) – True if width and points are given as resolution units instead of layout units.

**class** BPG.photonic\_objects.**PhotonicVia** (*tech, bbox, bot\_layer, top\_layer, bot\_dir, nx=1, ny=1, spx=0, spy=0, extend=True, top\_dir=None, unit\_mode=False*)

Bases: bag.layout.objects.Via

A layout via, with optional arraying parameters.

#### Parameters

- **tech** (*bag.layout.core.TechInfo*) – the technology class used to calculate via information.
- **bbox** (*bag.layout.util.BBox* or *bag.layout.util.BBoxArray*) – the via bounding box, not including extensions. If this is a BBoxArray, the BBoxArray's arraying parameters are used.
- **bot\_layer** (*str* or *(str, str)*) – the bottom layer name, or a tuple of layer name and purpose name. If purpose name not given, defaults to 'drawing'.
- **top\_layer** (*str* or *(str, str)*) – the top layer name, or a tuple of layer name and purpose name. If purpose name not given, defaults to 'drawing'.
- **bot\_dir** (*str*) – the bottom layer extension direction. Either 'x' or 'y'.
- **nx** (*int*) – arraying parameter. Number of columns.



- **ny** (*int*) – arraying parameter. Mumber of rows.
- **spx** (*float*) – arraying parameter. Column pitch.
- **spy** (*float*) – arraying parameter. Row pitch.
- **extend** (*bool*) – True if via extension can be drawn outside of bounding box.
- **top\_dir** (*Optional[str]*) – top layer extension direction. Can force to extend in same direction as bottom.
- **unit\_mode** (*bool*) – True if array pitches are given in resolution units.

**classmethod from\_content** (*content*)

**class** BPG.photonic\_objects.**PhotonicViaInfo** (*res, \*\*kwargs*)

Bases: bag.layout.objects.ViaInfo

A dictionary that represents a layout via.

**param\_list** = ['id', 'loc', 'orient', 'num\_rows', 'num\_cols', 'sp\_rows', 'sp\_cols', 'en

### 1.1.11 BPG.photonic\_port module

**class** BPG.photonic\_port.**PhotonicPort** (*name, center, orientation, width, layer, resolution, unit\_mode=False*)

Bases: object

**center**

Return the center coordinates as np array

**center\_unit**

Return the center coordinates as np array in resolution units

**classmethod from\_dict** (*center, name, orient, port\_width, layer, resolution, unit\_mode=True*)

Creates a new PhotonicPort object from a set of arguments

#### Parameters

- **center** (*Tuple[Union[float, int], Union[float, int]]*) – the (x, y) point of the port
- **name** (*str*) – the name of the port
- **orient** (*str*) – the orientation pointing into the object of the port
- **port\_width** (*Union[float, int]*) – the port width
- **layer** (*Union[Tuple[str, str], str]*) – the layer / layer purpose pair on which the port should be drawn. If the purpose is not specified, it is defaulted to the 'port' purpose
- **resolution** (*float*) – the grid resolution
- **unit\_mode** (*bool*) – True if layout dimensions are specified in resolution units

**Returns** **port** – the generated port

**Return type** *PhotonicPort*

**is\_horizontal** ()

Returns True if port orientation is R0 or R180

**is\_vertical** ()

Returns True if port orientation is vertical (R90 or R270)

**layer**

Returns the layer of the port

**name**

Returns the name of the port

**orientation**

Returns the orientation of the port

**resolution**

Returns the layout resolution of the port object

**transform** (*loc*=(0, 0), *orient*='R0', *unit\_mode*=False)

Return a new transformed photonic port

**Parameters**

- **loc** (*Tuple*[*Union*[*float*, *int*], *Union*[*float*, *int*]]) – the x, y coordinate to move the port
- **orient** (*str*) – the orientation to rotate the port
- **unit\_mode** (*bool*) – true if layout dimensions are specified in resolution units

**Returns** **port** – the transformed photonic port object

**Return type** *PhotonicPort*

**used**

Returns True if port is used

**width**

Returns the width of the port

**width\_unit**

Returns the width of the port in layout units

**width\_vec** (*unit\_mode*=True, *normalized*=True)

Returns a normalized vector pointing into the port object

**Parameters**

- **unit\_mode** (*bool*) – True to return vector in resolution units
- **normalized** (*bool*) – True to normalize the vector. If False, vector magnitude is the port width

**Returns** **vec** – a vector whos orientation points into the port and whos magnitude is either 1 or the waveguide port width

**Return type** *np.array*

## 1.1.12 BPG.photonic\_template module

```
class BPG.photonic_template.PhotonicTemplateBase (temp_db, lib_name, params,  
                                                used_names, **kwargs)
```

Bases: *bag.layout.template.TemplateBase*

```
add_advancedpolygon (polygon)
```

```
add_instance (master, inst_name=None, loc=(0, 0), orient='R0', nx=1, ny=1, spx=0, spy=0,  
              unit_mode=False)
```

Adds a new (arrayed) instance to layout.

**Parameters**

- **master** (*TemplateBase*) – the master template object.
- **inst\_name** (*Optional[str]*) – instance name. If None or an instance with this name already exists, a generated unique name is used.
- **loc** (*Tuple[Union[float, int], Union[float, int]]*) – instance location.
- **orient** (*str*) – instance orientation. Defaults to “R0”
- **nx** (*int*) – number of columns. Must be positive integer.
- **ny** (*int*) – number of rows. Must be positive integer.
- **spx** (*Union[float, int]*) – column pitch. Used for arraying given instance.
- **spy** (*Union[float, int]*) – row pitch. Used for arraying given instance.
- **unit\_mode** (*bool*) – True if dimensions are given in resolution units.

**Returns** *inst* – the added instance.

**Return type** *Instance*

**add\_instances\_port\_to\_port** (*inst\_master*, *instance\_port\_name*, *self\_port=None*, *self\_port\_name=None*, *instance\_name=None*, *reflect=False*)

Instantiates a new instance of the *inst\_master* template. The new instance is placed such that its port named ‘*instance\_port\_name*’ is aligned-with and touching the ‘*self\_port*’ or ‘*self\_port\_name*’ port of the current hierarchy level.

The new instance is rotated about the new instance’s master’s origin until desired port is aligned. Optional reflection is performed after rotation, about the port axis.

The self port being connected to can be specified either by passing a *self\_port* PhotonicPort object, or by passing the *self\_port\_name*, which refers to a port that must exist in the current hierarchy level.

**Parameters**

- **inst\_master** (*PhotonicTemplateBase*) – the template master to be added
- **instance\_port\_name** (*str*) – the name of the port in the added instance to connect to
- **self\_port** (*Optional[PhotonicPort]*) – the photonic port object in the current hierarchy to connect to. Has priority over *self\_port\_name*
- **self\_port\_name** (*Optional[str]*) – the name of the port in the current hierarchy to connect to
- **instance\_name** (*Optional[str]*) – the name to give the new instance
- **reflect** (*bool*) – True to flip the added instance after rotation

**Returns** *new\_inst* – the newly added instance

**Return type** *PhotonicInstance*

**add\_monitor\_obj** (*monitor\_obj*)

Add a new Lumerical monitor object to the db

**add\_path** (*path*)

Adds a PhotonicPath to the layout object

**Parameters** *path* (*PhotonicPath*) –

**Returns** *path*

Return type *PhotonicPath*

**add\_photonic\_port** (*name=None, center=None, orient=None, width=None, layer=None, resolution=None, unit\_mode=False, port=None, overwrite=False, show=True*)

Adds a photonic port to the current hierarchy. A PhotonicPort object can be passed, or will be constructed if the proper arguments are passed to this function.

#### Parameters

- **name** (*str*) – name to give the new port
- **center** (*coord\_type*) – (x, y) location of the port
- **orient** (*str*) – orientation pointing INTO the port
- **width** (*dim\_type*) – the port width
- **layer** (*Union[str, Tuple[str, str]]*) – the layer on which the port should be added. If only a string, the purpose is defaulted to ‘port’
- **resolution** (*Union[float, int]*) – the grid resolution
- **unit\_mode** (*bool*) – True if layout dimensions are specified in resolution units
- **port** (*Optional[PhotonicPort]*) – the PhotonicPort object to add. This argument can be provided in lieu of all the others.
- **overwrite** (*bool*) – True to add the port with the specified name even if another port with that name already exists in this level of the design hierarchy.
- **show** (*bool*) – True to draw the port indicator shape

Returns **port** – the added photonic port object

Return type *PhotonicPort*

**add\_polygon** (*polygon=None, layer=None, points=None, resolution=None, unit\_mode=False*)

Add a polygon to the layout. If photonic polygon object is passed, use it. User can also pass information to create a new photonic polygon.

#### Parameters

- **polygon** (*Optional[PhotonicPolygon]*) – the polygon to add
- **layer** (*Union[str, Tuple[str, str]]*) – the layer of the polygon
- **resolution** (*float*) – the layout grid resolution
- **points** (*List[coord\_type]*) – the points defining the polygon
- **unit\_mode** (*bool*) – True if the points are given in resolution units

Returns **polygon** – the added polygon object

Return type *PhotonicPolygon*

**add\_rect** (*layer, x\_span=None, y\_span=None, center=None, coord1=None, coord2=None, bbox=None, nx=1, ny=1, spx=0, spy=0, unit\_mode=False*)

Add a new (arrayed) rectangle.

#### Parameters

- **layer** (*Union[str, Tuple[str, str]]*) – the layer name, or the (layer, purpose) pair.
- **x\_span** (*Union[int, float]*) – horizontal span of the rectangle.
- **y\_span** (*Union[int, float]*) – vertical span of the rectangle.

- **center** (*Union[int, float]*) – coordinate defining center point of the rectangle.
- **coord1** (*Tuple[Union[int, float], Union[int, float]]*) – point defining one corner of rectangle boundary.
- **coord2** (*Tuple[Union[int, float], Union[int, float]]*) – opposite corner from coord1 defining rectangle boundary.
- **bbox** (*bag.layout.util.BBox or bag.layout.util.BBoxArray*) – the base bounding box. If this is a BBoxArray, the BBoxArray's arraying parameters are used.
- **nx** (*int*) – number of columns.
- **ny** (*int*) – number of rows.
- **spx** (*float*) – column pitch.
- **spy** (*float*) – row pitch.
- **unit\_mode** (*bool*) – True if layout dimensions are specified in resolution units.

**Returns** **rect** – the added rectangle.

**Return type** *PhotonicRect*

**add\_round** (*round\_obj*)

**Parameters** **round\_obj** (*Optional[PhotonicRound]*) – the polygon to add

**Returns** **polygon** – the added round object

**Return type** *PhotonicRound*

**add\_sim\_obj** (*sim\_obj*)

Add a new Lumerical simulation object to the db

**add\_source\_obj** (*source\_obj*)

Add a new Lumerical source object to the db

**add\_via\_stack** (*bot\_layer, top\_layer, loc, min\_area\_on\_bot\_top\_layer=False, unit\_mode=False*)

Adds a via stack with one via in each layer at the provided location. All intermediate layers will be enclosed with an enclosure that satisfies both via rules and min area rules

**Parameters**

- **bot\_layer** (*str*) – Name of the bottom layer
- **top\_layer** (*str*) – Name of the top layer
- **loc** – (x, y) location of the center of the via stack
- **min\_area\_on\_bot\_top\_layer** (*bool*) – True to have enclosures on top and bottom layer satisfy minimum area constraints
- **unit\_mode** – True if input argument is specified in layout resolution units

**add\_via\_stack\_by\_ind** (*bot\_layer\_ind, top\_layer\_ind, loc, min\_area\_on\_bot\_top\_layer=False, unit\_mode=False*)

**delete\_port** (*port\_names*)

Removes the given ports from this instances list of ports. Raises error if given port does not exist.

**Parameters** **port\_names** (*Union[str, List[str]]*) –

**draw\_layout()**

Draw the layout of this template.

Override this method to create the layout.

WARNING: you should never call this method yourself.

**extract\_photonic\_ports** (*inst*, *port\_names=None*, *port\_renaming=None*, *unmatched\_only=True*,  
*show=True*)

Brings ports from lower level of hierarchy to the current hierarchy level

**Parameters**

- **inst** (*PhotonicInstance*) – the instance that contains the ports to be extracted
- **port\_names** (*Optional[Union[str, List[str]]*) – the port name or list of port names re-export. If not supplied, all ports of the inst will be extracted
- **port\_renaming** (*Optional[Dict[str, str]]*) – a dictionary containing key-value pairs mapping inst's port names (key) to the new desired port names (value). If not supplied, extracted ports will be given their original names
- **unmatched\_only** (*bool*) –
- **show** (*bool*) –

**finalize()**

**get\_photonic\_port** (*port\_name=""*)

Returns the photonic port object with the given name

**Parameters** **port\_name** (*Optional[str]*) – the photonic port terminal name. If None or empty, check if this photonic template has only one port, and return it

**Returns** **port** – The photonic port object

**Return type** *PhotonicPort*

**has\_photonic\_port** (*port\_name*)

Checks if the given port name exists in the current hierarchy level.

**Parameters** **port\_name** (*str*) – the name of the port

**Returns**

- *boolean*
- *true if port exists in current hierarchy level*

**photonic\_ports\_names\_iter()**

**update\_port()**

```
class BPG.photonic_template.PhotonicTemplateDB (lib_defs,          routing_grid,          lib-  
name, prj=None, name_prefix="",  
name_suffix="", use_cybagoa=False,  
gds_lay_file="", flatten=False,  
gds_filepath="", lsf_filepath="", dat-  
aprep_file="", lsf_export_filepath="",  
**kwargs)
```

Bases: bag.layout.template.TemplateDB

**by\_layer\_polygon\_list\_to\_flat\_for\_gds\_export()**

Converts a LPP-keyed dictionary of polygon pointlists to a flat content list format for GDS export

**dataprep** (*dataprep\_file: str, push\_portshapes\_through\_dataprep: bool = False, debug: bool = False*)  
 → None

Takes the flat content list and performs the specified transformations on the shapes for the purpose of cleaning DRC and prepping tech specific functions.

## Notes

1. Take the shapes in the flattened content list and convert them to gdspy format
2. Parse the dataprep spec file to extract the desired procedure defined through dataprep\_groups
- 3) Perform each dataprep operation on the provided layers in order. dataprep\_groups is a list where each element contains 2 other lists:
  - 3a) lpp\_in defines the layers that the operation will be performed on
  - 3b) lpp\_ops defines the operation to be performed
  - 3c) Maps the operation in the spec file to its gdspy implementation and performs it
4. Performs a final over\_under\_under\_over operation
5. Take the dataprepped gdspy shapes and import them into a new post-dataprep content list

## Parameters

- **dataprep\_file** (*str*) – path to yaml containing dataprep procedure
- **debug** (*bool*) – True to print debug information
- **push\_portshapes\_through\_dataprep** (*bool*) – True to perform dataprep and convert the port indicator shapes

**generate\_flat\_content\_list\_from\_dataprep** (*poly\_list\_by\_layer, sim\_obj\_list*)

Takes the output of dataprep and converts it into a flat content list

## Parameters

- **poly\_list\_by\_layer** (*Dict[Str, List]*) – A dictionary containing lists all dataprepped polygons organized by layername
- **sim\_obj\_list** (*Tuple[List, List, List]*) – A tuple of lists containing all simulation objects to be used

**get\_content\_on\_layer** (*layer*)

Returns only the content that exists on a given layer

**Parameters** **layer** (*Tuple[str, str]*) – the layer whose content is desired

**Returns** **content** – the shape content on the provided layer

**Return type** Tuple

**get\_polygon\_point\_lists\_on\_layer** (*layer, debug=False*)

Returns a list of all shapes

## Parameters

- **layer** (*Tuple[str, str]*) – the layer purpose pair to get all shapes in shapely format
- **debug** (*bool*) – true to print debug info

**instantiate\_flat\_masters** (*master\_list*, *name\_list*=None, *lib\_name*=", *debug*=False, *rename\_dict*=None, *draw\_flat\_gds*=True, *sort\_by\_layer*=True) → None

Create all given masters in the database to a flat hierarchy.

#### Parameters

- **master\_list** (*Sequence*[*DesignMaster*]) – list of masters to instantiate.
- **name\_list** (*Optional*[*Sequence*[*Optional*[*str*]]]) – list of master cell names. If not given, default names will be used.
- **lib\_name** (*str*) – Library to create the masters in. If empty or None, use default library.
- **debug** (*bool*) – True to print debugging messages
- **rename\_dict** (*Optional*[*Dict*[*str*, *str*]]) – optional master cell renaming dictionary.

**instantiate\_masters** (*master\_list*, *name\_list*=None, *lib\_name*=", *debug*=False, *rename\_dict*=None) → None

Create all given masters in the database. Currently, this is being overridden so that the *content\_list* is stored locally. This is a little hacky, and may need to be changed pending further testing

#### Parameters

- **master\_list** (*Sequence*[*DesignMaster*]) – list of masters to instantiate.
- **name\_list** (*Optional*[*Sequence*[*Optional*[*str*]]]) – list of master cell names. If not given, default names will be used.
- **lib\_name** (*str*) – Library to create the masters in. If empty or None, use default library.
- **debug** (*bool*) – True to print debugging messages
- **rename\_dict** (*Optional*[*Dict*[*str*, *str*]]) – optional master cell renaming dictionary.

**lsf\_dataprep** (*dataprep\_file*: *str*, *push\_portshapes\_through\_dataprep*: *bool* = False, *debug*: *bool* = False) → None

Takes the flat content list and prepares the shapes to be exported to lumerical.

## Notes

1. Take the shapes in the flattened content list and convert them to gdspy format
2. Parse the dataprep spec file to extract the desired procedure defined through *dataprep\_groups*
- 3) Perform each dataprep operation on the provided layers in order. *dataprep\_groups* is a list where each element contains 2 other lists:
  - 3a) *lpp\_in* defines the layers that the operation will be performed on
  - 3b) *lpp\_ops* defines the operation to be performed
  - 3c) Maps the operation in the spec file to its gdspy implementation and performs it
4. Performs a final over\_under\_under\_over operation
5. Take the dataprepped gdspy shapes and import them into a new post-dataprep content list

#### Parameters

- **dataprep\_file** (*str*) – path to yaml containing dataprep procedure



- **debug** (*bool*) – True to print debug information
- **push\_portshapes\_through\_dataprep** (*bool*) – True to perform dataprep and convert the port indicator shapes

**sort\_flat\_content\_by\_layers** ()

Sorts the flattened content list into a dictionary of content lists, with keys corresponding to a given lpp

**to\_lumerical** (*gds\_layermap: str, lsf\_export\_config: str, lsf\_filepath: str, debug: bool = False*) →

None

Exports shapes into the lumerical LSF format

## Notes

1. Import tech information for the layermap and lumerical properties
2. Make sure that a flat content list has been generated for the layout already
3. If dataprep is called, run the procedure in the lsf\_export\_config
4. For each element in the flat content list, convert it into lsf code and append to running export file
5. Lsf code is generated by sending properties and tech info to the lsf\_export static method in each shape class
6. Lsf code is appended to the running file with LumericalDesignGenerator

### Parameters

- **gds\_layermap** (*str*) – path to yaml containing tech specific gds layer information
- **lsf\_export\_config** (*str*) – path to yaml containing lumerical export configurations
- **lsf\_filepath** (*str*) – path to where new lsf will be created
- **debug** (*bool*) – True to display profiling information

**to\_polygon\_pointlist\_from\_content\_list** (*content\_list, debug=False*)

Convert the provided content list into two lists of polygon pointlists. The first returned list represents the positive boundaries of polygons. The second returned list represents the ‘negative’ boundaries of holes in polygons. All shapes in the passed content list are converted, regardless of layer. It is expected that the content list passed to this function only has a single LPP’s content

### Parameters

- **content\_list** (*List*) – The content list to be converted to a polygon pointlist
- **debug** (*bool*) – True to print debug information

**Returns positive\_polygon\_pointlist, negative\_polygon\_pointlist** – The positive shape and negative shape (holes) polygon boundaries

**Return type** Tuple[List, List]

## 1.1.13 BPG.poly\_simplify module

**BPG.poly\_simplify.coord\_to\_shapely** (*pos\_neg\_list\_list*)

Converts list of coordinate lists into shapely polygon objects

**Parameters** *pos\_neg\_list\_list* –

`BPG.poly_simplify.shapely_to_gdspy(geom_shapely)`

`BPG.poly_simplify.shapely_to_gdspy_polygon(polygon_shapely)`

`BPG.poly_simplify.simplify_coord_to_gdspy(pos_neg_list_list, tolerance=0.0005)`

### 1.1.14 BPG.test\_setup module

`BPG.test_setup.bpg_setup()`

Creates the BAG project instance to be used

### 1.1.15 Module contents

## PYTHON MODULE INDEX

### b

BPG, [30](#)  
BPG.dataprep\_gdspy, [1](#)  
BPG.lumerical\_generator, [5](#)  
BPG.lumerical\_sim, [6](#)  
BPG.lumerical\_tb, [7](#)  
BPG.manh\_shapely, [8](#)  
BPG.photonic\_core, [9](#)  
BPG.photonic\_layout\_manager, [12](#)  
BPG.photonic\_objects, [13](#)  
BPG.photonic\_port, [21](#)  
BPG.photonic\_template, [22](#)  
BPG.poly\_simplify, [29](#)  
BPG.test\_setup, [30](#)



# INDEX

## A

- add\_advancedpolygon() (BPG.photonic\_template.PhotonicTemplateBase method), 22
- add\_code() (BPG.lumerical\_generator.LumericalCodeGenerator method), 5
- add\_code\_block() (BPG.lumerical\_generator.LumericalCodeGenerator method), 5
- add\_effective\_index\_monitor() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_EME\_port() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_eme\_profile() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_EME\_solver() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_FDE\_solver() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_FDTD\_port() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_FDTD\_solver() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_freq\_domain\_monitor() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_gaussian\_source() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_index\_monitor() (BPG.lumerical\_tb.LumericalTB method), 7
- add\_instance() (BPG.photonic\_template.PhotonicTemplateBase method), 22
- add\_instances\_port\_to\_port() (BPG.photonic\_template.PhotonicTemplateBase method), 23
- add\_line() (BPG.lumerical\_generator.LumericalCodeGenerator method), 6
- add\_mode\_expansion\_monitor() (BPG.lumerical\_tb.LumericalTB method), 8
- add\_mode\_source() (BPG.lumerical\_tb.LumericalTB method), 8
- add\_monitor\_obj() (BPG.photonic\_core.PhotonicBagLayout method), 10
- add\_monitor\_obj() (BPG.photonic\_template.PhotonicTemplateBase method), 23
- add\_movie\_monitor() (BPG.lumerical\_tb.LumericalTB method), 8
- add\_path() (BPG.photonic\_template.PhotonicTemplateBase method), 23
- add\_photonic\_port() (BPG.photonic\_template.PhotonicTemplateBase method), 24
- add\_point\_source() (BPG.lumerical\_tb.LumericalTB method), 8
- add\_polygon() (BPG.photonic\_template.PhotonicTemplateBase method), 24
- add\_rect() (BPG.photonic\_template.PhotonicTemplateBase method), 24
- add\_round() (BPG.photonic\_core.PhotonicBagLayout method), 10
- add\_round() (BPG.photonic\_template.PhotonicTemplateBase method), 25
- add\_sim\_obj() (BPG.photonic\_core.PhotonicBagLayout method), 10
- add\_sim\_obj() (BPG.photonic\_template.PhotonicTemplateBase method), 25
- add\_source\_obj() (BPG.photonic\_core.PhotonicBagLayout method), 10
- add\_source\_obj() (BPG.photonic\_template.PhotonicTemplateBase method), 25
- add\_sweep\_point() (BPG.lumerical\_generator.LumericalSweepGenerator method), 6
- add\_time\_domain\_monitor() (BPG.lumerical\_tb.LumericalTB method), 8
- add\_total\_field\_source() (BPG.lumerical\_tb.LumericalTB method), 8
- add\_var\_FDTD\_solver() (BPG.lumerical\_tb.LumericalTB method), 8
- add\_via\_stack() (BPG.photonic\_template.PhotonicTemplateBase method), 25
- add\_via\_stack\_by\_ind() (BPG.photonic\_template.PhotonicTemplateBase method), 25
- align\_to\_port() (BPG.lumerical\_sim.FDESolver method), 6

`align_to_port()` (BPG.photonic\_core.Box method), 9

## B

Box (class in BPG.photonic\_core), 9

BPG (module), 30

BPG.dataprep\_gdspsy (module), 1

BPG.lumerical\_generator (module), 5

BPG.lumerical\_sim (module), 6

BPG.lumerical\_tb (module), 7

BPG.manh\_shapely (module), 8

BPG.photonic\_core (module), 9

BPG.photonic\_layout\_manager (module), 12

BPG.photonic\_objects (module), 13

BPG.photonic\_port (module), 21

BPG.photonic\_template (module), 22

BPG.poly\_simplify (module), 29

BPG.test\_setup (module), 30

`bpg_setup()` (in module BPG.test\_setup), 30

`by_layer_polygon_list_to_flat_for_gds_export()`  
(BPG.photonic\_template.PhotonicTemplateDB  
method), 26

## C

`center` (BPG.photonic\_objects.PhotonicRound attribute),  
19

`center` (BPG.photonic\_port.PhotonicPort attribute), 21

`center_unit` (BPG.photonic\_objects.PhotonicRound at-  
tribute), 19

`center_unit` (BPG.photonic\_port.PhotonicPort attribute),  
21

`cleanup_loop()` (in module BPG.dataprep\_gdspsy), 1

`cleanup_loop()` (in module BPG.manh\_shapely), 8

`construct_tb()` (BPG.lumerical\_tb.LumericalTB method),  
8

`content` (BPG.lumerical\_sim.LumericalSimObj attribute),  
7

`content` (BPG.photonic\_objects.PhotonicInstance at-  
tribute), 14

`content` (BPG.photonic\_objects.PhotonicPath attribute),  
15

`content` (BPG.photonic\_objects.PhotonicRound at-  
tribute), 19

`coord_to_shapely()` (in module BPG.dataprep\_gdspsy), 1

`coord_to_shapely()` (in module BPG.poly\_simplify), 29

CoordBase (class in BPG.photonic\_core), 9

`coords_aprx_in_line()` (in module  
BPG.dataprep\_gdspsy), 1

`coords_aprx_in_line()` (in module BPG.manh\_shapely),  
8

`coords_cleanup()` (in module BPG.dataprep\_gdspsy), 1

`coords_cleanup()` (in module BPG.manh\_shapely), 8

`copy()` (BPG.photonic\_objects.PhotonicInstanceInfo  
method), 15

`create_dut()` (BPG.lumerical\_tb.LumericalTB method), 8

`create_sweep_loop()` (BPG.lumerical\_generator.LumericalSweepGenerator  
method), 6

## D

`dataprep()` (BPG.photonic\_layout\_manager.PhotonicLayoutManager  
method), 12

`dataprep()` (BPG.photonic\_template.PhotonicTemplateDB  
method), 26

`dataprep_cleanup_gdspsy()` (in module  
BPG.dataprep\_gdspsy), 2

`dataprep_coord_to_gdspsy()` (in module  
BPG.dataprep\_gdspsy), 2

`dataprep_oversize_gdspsy()` (in module  
BPG.dataprep\_gdspsy), 2

`dataprep_roughsize_gdspsy()` (in module  
BPG.dataprep\_gdspsy), 2

`dataprep_undersize_gdspsy()` (in module  
BPG.dataprep\_gdspsy), 3

`delete_port()` (BPG.photonic\_template.PhotonicTemplateBase  
method), 25

`draw_layout()` (BPG.lumerical\_tb.LumericalTB method),  
8

`draw_layout()` (BPG.photonic\_template.PhotonicTemplateBase  
method), 25

## E

`export_to_lsf()` (BPG.lumerical\_generator.LumericalDesignGenerator  
method), 6

`export_to_lsf()` (BPG.lumerical\_generator.LumericalSweepGenerator  
method), 6

`extract_photonic_ports()` (BPG.photonic\_template.PhotonicTemplateBase  
method), 26

## F

FDESolver (class in BPG.lumerical\_sim), 6

FDTD Solver (class in BPG.lumerical\_sim), 7

`finalize()` (BPG.photonic\_core.PhotonicBagLayout  
method), 10

`finalize()` (BPG.photonic\_template.PhotonicTemplateBase  
method), 26

`finalize_template()` (BPG.photonic\_core.PTech method),  
10

float (BPG.photonic\_core.CoordBase attribute), 9

`from_content()` (BPG.photonic\_objects.PhotonicBlockage  
class method), 13

`from_content()` (BPG.photonic\_objects.PhotonicBoundary  
class method), 14

`from_content()` (BPG.photonic\_objects.PhotonicPath  
class method), 15

`from_content()` (BPG.photonic\_objects.PhotonicPinInfo  
class method), 17

`from_content()` (BPG.photonic\_objects.PhotonicPolygon  
class method), 17

- from\_content() (BPG.photonic\_objects.PhotonicRect class method), 18
- from\_content() (BPG.photonic\_objects.PhotonicRound class method), 19
- from\_content() (BPG.photonic\_objects.PhotonicVia class method), 21
- from\_dict() (BPG.photonic\_port.PhotonicPort class method), 21
- ## G
- gdspy\_manh() (in module BPG.dataprep\_gdspy), 3
- generate\_flat\_content\_list\_from\_dataprep() (BPG.photonic\_template.PhotonicTemplateDB method), 27
- generate\_flat\_gds() (BPG.photonic\_layout\_manager.PhotonicLayoutManager method), 12
- generate\_gds() (BPG.photonic\_layout\_manager.PhotonicLayoutManager method), 12
- generate\_lsf() (BPG.photonic\_layout\_manager.PhotonicLayoutManager method), 13
- generate\_tb() (BPG.photonic\_layout\_manager.PhotonicLayoutManager method), 13
- get\_bound\_box\_of() (BPG.photonic\_objects.PhotonicInstance class method), 14
- get\_content() (BPG.photonic\_core.PhotonicBagLayout method), 10
- get\_content\_on\_layer() (BPG.photonic\_template.PhotonicTemplateDB method), 27
- get\_file\_header() (BPG.lumerical\_generator.LumericalCodeGenerator method), 6
- get\_layer\_id() (BPG.photonic\_core.PTech method), 10
- get\_params\_info() (BPG.lumerical\_tb.LumericalTB class method), 8
- get\_photonic\_port() (BPG.photonic\_objects.PhotonicInstance class method), 14
- get\_photonic\_port() (BPG.photonic\_template.PhotonicTemplateDB method), 26
- get\_polygon\_point\_lists\_on\_layer() (BPG.photonic\_template.PhotonicTemplateDB method), 27
- get\_port\_used() (BPG.photonic\_objects.PhotonicInstance class method), 15
- ## H
- has\_photonic\_port() (BPG.photonic\_template.PhotonicTemplateDB method), 26
- ## I
- instantiate\_flat\_masters() (BPG.photonic\_template.PhotonicTemplateDB method), 27
- instantiate\_masters() (BPG.photonic\_template.PhotonicTemplateDB method), 28
- is\_horizontal() (BPG.photonic\_port.PhotonicPort method), 21
- is\_vertical() (BPG.photonic\_port.PhotonicPort method), 21
- ## L
- layer (BPG.photonic\_objects.PhotonicPath attribute), 15
- layer (BPG.photonic\_objects.PhotonicRound attribute), 19
- layer (BPG.photonic\_port.PhotonicPort attribute), 21
- load\_yaml() (BPG.photonic\_core.PhotonicBagProject static method), 11
- load\_yaml() (BPG.photonic\_layout\_manager.PhotonicLayoutManager static method), 13
- lower (BPG.photonic\_objects.PhotonicPath attribute), 16
- lsf\_dataprep() (BPG.photonic\_template.PhotonicTemplateDB method), 28
- lsf\_export() (BPG.lumerical\_sim.FDESolver method), 6
- lsf\_export() (BPG.lumerical\_sim.FDTD solver method), 7
- lsf\_export() (BPG.lumerical\_sim.LumericalSimObj method), 7
- lsf\_export() (BPG.photonic\_objects.PhotonicPolygon class method), 17
- lsf\_export() (BPG.photonic\_objects.PhotonicRect class method), 18
- lsf\_export() (BPG.photonic\_objects.PhotonicRound class method), 19
- LumericalCodeGenerator (class in BPG.lumerical\_generator), 5
- LumericalDesignGenerator (class in BPG.lumerical\_generator), 6
- LumericalSimObj (class in BPG.lumerical\_sim), 7
- LumericalSweepGenerator (class in BPG.lumerical\_generator), 6
- LumericalTB (class in BPG.lumerical\_tb), 7
- ## M
- make\_tdb() (BPG.photonic\_layout\_manager.PhotonicLayoutManager method), 13
- manh\_skill() (in module BPG.dataprep\_gdspy), 3
- manh\_skill() (in module BPG.manh\_shapely), 8
- master (BPG.photonic\_objects.PhotonicInstance attribute), 15
- master\_key (BPG.photonic\_objects.PhotonicInstanceInfo attribute), 15
- mesh\_size (BPG.lumerical\_sim.FDESolver attribute), 7
- meters (BPG.photonic\_core.CoordBase attribute), 9
- micron (BPG.photonic\_core.CoordBase attribute), 10
- microns (BPG.photonic\_core.CoordBase attribute), 10
- move\_all\_by() (BPG.photonic\_core.PhotonicBagLayout method), 10
- move\_by() (BPG.photonic\_core.Box method), 9

- move\_by() (BPG.photonic\_objects.PhotonicInstance method), 15
- move\_by() (BPG.photonic\_objects.PhotonicPath method), 16
- move\_by() (BPG.photonic\_objects.PhotonicRound method), 19
- ## N
- name (BPG.photonic\_port.PhotonicPort attribute), 22
- not\_manh() (in module BPG.dataprep\_gdspy), 4
- num\_of\_sparse\_point\_round() (BPG.photonic\_objects.PhotonicRound static method), 19
- ## O
- orientation (BPG.lumerical\_sim.FDESolver attribute), 7
- orientation (BPG.photonic\_port.PhotonicPort attribute), 22
- ## P
- param\_list (BPG.photonic\_objects.PhotonicPinInfo attribute), 17
- param\_list (BPG.photonic\_objects.PhotonicViaInfo attribute), 21
- photonic\_ports\_names\_iter() (BPG.photonic\_template.PhotonicTemplateBase method), 26
- PhotonicAdvancedPolygon (class in BPG.photonic\_objects), 13
- PhotonicBagLayout (class in BPG.photonic\_core), 10
- PhotonicBagProject (class in BPG.photonic\_core), 11
- PhotonicBlockage (class in BPG.photonic\_objects), 13
- PhotonicBoundary (class in BPG.photonic\_objects), 13
- PhotonicInstance (class in BPG.photonic\_objects), 14
- PhotonicInstanceInfo (class in BPG.photonic\_objects), 15
- PhotonicLayoutManager (class in BPG.photonic\_layout\_manager), 12
- PhotonicPath (class in BPG.photonic\_objects), 15
- PhotonicPathCollection (class in BPG.photonic\_objects), 16
- PhotonicPinInfo (class in BPG.photonic\_objects), 16
- PhotonicPolygon (class in BPG.photonic\_objects), 17
- PhotonicPort (class in BPG.photonic\_port), 21
- PhotonicRect (class in BPG.photonic\_objects), 17
- PhotonicRound (class in BPG.photonic\_objects), 18
- PhotonicTemplateBase (class in BPG.photonic\_template), 22
- PhotonicTemplateDB (class in BPG.photonic\_template), 26
- PhotonicTLineBus (class in BPG.photonic\_objects), 20
- PhotonicVia (class in BPG.photonic\_objects), 20
- PhotonicViaInfo (class in BPG.photonic\_objects), 21
- Plane (class in BPG.photonic\_core), 11
- plane\_wave\_source() (BPG.lumerical\_tb.LumericalTB method), 8
- plot\_coords() (in module BPG.manh\_shapely), 9
- plot\_line() (in module BPG.manh\_shapely), 9
- points (BPG.photonic\_objects.PhotonicPath attribute), 16
- points\_unit (BPG.photonic\_objects.PhotonicPath attribute), 16
- poly\_operation() (in module BPG.dataprep\_gdspy), 4
- polygon\_pointlist\_export() (BPG.photonic\_objects.PhotonicPath class method), 16
- polygon\_pointlist\_export() (BPG.photonic\_objects.PhotonicPolygon class method), 17
- polygon\_pointlist\_export() (BPG.photonic\_objects.PhotonicRect class method), 18
- polygon\_pointlist\_export() (BPG.photonic\_objects.PhotonicRound class method), 19
- polygon\_points (BPG.photonic\_objects.PhotonicPath attribute), 16
- polyop\_gdspy\_to\_point\_list() (in module BPG.dataprep\_gdspy), 4
- polyop\_manh() (in module BPG.manh\_shapely), 9
- polyop\_manh\_polygon() (in module BPG.manh\_shapely), 9
- process\_points() (BPG.photonic\_objects.PhotonicPath method), 16
- PTech (class in BPG.photonic\_core), 10
- ## R
- res (BPG.photonic\_core.CoordBase attribute), 10
- resolution (BPG.photonic\_port.PhotonicPort attribute), 22
- rin (BPG.photonic\_objects.PhotonicRound attribute), 19
- rin\_unit (BPG.photonic\_objects.PhotonicRound attribute), 19
- rout (BPG.photonic\_objects.PhotonicRound attribute), 19
- rout\_unit (BPG.photonic\_objects.PhotonicRound attribute), 19
- ## S
- set() (BPG.lumerical\_generator.LumericalCodeGenerator method), 6
- set\_center\_span() (BPG.photonic\_core.Box method), 9
- set\_port\_used() (BPG.photonic\_objects.PhotonicInstance method), 15
- set\_span() (BPG.photonic\_core.Box method), 9
- shapely\_to\_gdspy() (in module BPG.dataprep\_gdspy), 5
- shapely\_to\_gdspy() (in module BPG.poly\_simplify), 29
- shapely\_to\_gdspy\_polygon() (in module BPG.dataprep\_gdspy), 5



shapely\_to\_gdsapy\_polygon() (in module BPG.poly\_simplify), 30  
 simplify\_coord\_to\_gdsapy() (in module BPG.dataprep\_gdsapy), 5  
 simplify\_coord\_to\_gdsapy() (in module BPG.poly\_simplify), 30  
 sort\_flat\_content\_by\_layers() (BPG.photonic\_template.PhotonicTemplateDB method), 29

## T

theta0 (BPG.photonic\_objects.PhotonicRound attribute), 20  
 theta1 (BPG.photonic\_objects.PhotonicRound attribute), 20  
 to\_lumerical() (BPG.photonic\_template.PhotonicTemplateDB method), 29  
 to\_polygon\_pointlist\_from\_content\_list() (BPG.photonic\_template.PhotonicTemplateDB method), 29  
 transform() (BPG.photonic\_objects.PhotonicInstance method), 15  
 transform() (BPG.photonic\_objects.PhotonicPath method), 16  
 transform() (BPG.photonic\_objects.PhotonicPinInfo method), 17  
 transform() (BPG.photonic\_objects.PhotonicRound method), 20  
 transform() (BPG.photonic\_port.PhotonicPort method), 22

## U

unit\_mode (BPG.photonic\_core.CoordBase attribute), 10  
 update\_port() (BPG.photonic\_template.PhotonicTemplateBase method), 26  
 upper (BPG.photonic\_objects.PhotonicPath attribute), 16  
 use\_flip\_parity() (BPG.photonic\_core.PTech method), 10  
 used (BPG.photonic\_port.PhotonicPort attribute), 22

## V

valid (BPG.photonic\_objects.PhotonicPath attribute), 16  
 value (BPG.photonic\_core.CoordBase attribute), 10

## W

width (BPG.photonic\_objects.PhotonicPath attribute), 16  
 width (BPG.photonic\_port.PhotonicPort attribute), 22  
 width\_unit (BPG.photonic\_objects.PhotonicPath attribute), 16  
 width\_unit (BPG.photonic\_port.PhotonicPort attribute), 22  
 width\_vec() (BPG.photonic\_port.PhotonicPort method), 22

## X

x (BPG.photonic\_core.XY attribute), 11  
 x (BPG.photonic\_core.XYZ attribute), 11  
 x\_float (BPG.photonic\_core.XY attribute), 11  
 x\_float (BPG.photonic\_core.XYZ attribute), 11  
 x\_meters (BPG.photonic\_core.XY attribute), 11  
 x\_meters (BPG.photonic\_core.XYZ attribute), 11  
 xy (BPG.photonic\_core.XY attribute), 11  
 XY (class in BPG.photonic\_core), 11  
 xy\_float (BPG.photonic\_core.XY attribute), 11  
 xy\_meters (BPG.photonic\_core.XY attribute), 11  
 xyz (BPG.photonic\_core.XYZ attribute), 11  
 XYZ (class in BPG.photonic\_core), 11  
 xyz\_float (BPG.photonic\_core.XYZ attribute), 11  
 xyz\_meters (BPG.photonic\_core.XYZ attribute), 12

## Y

y (BPG.photonic\_core.XY attribute), 11  
 y (BPG.photonic\_core.XYZ attribute), 12  
 y\_float (BPG.photonic\_core.XY attribute), 11  
 y\_float (BPG.photonic\_core.XYZ attribute), 12  
 y\_meters (BPG.photonic\_core.XY attribute), 11  
 y\_meters (BPG.photonic\_core.XYZ attribute), 12

## Z

z (BPG.photonic\_core.XYZ attribute), 12  
 z\_float (BPG.photonic\_core.XYZ attribute), 12  
 z\_meters (BPG.photonic\_core.XYZ attribute), 12